

For Reference

NOT TO BE TAKEN FROM THIS ROOM

For Reference

NOT TO BE TAKEN FROM THIS ROOM

Ex LIBRIS
UNIVERSITATIS
ALBERTAENSIS



Regulations Regarding Theses and Dissertations

[illegible]



Digitized by the Internet Archive
in 2019 with funding from
University of Alberta Libraries

<https://archive.org/details/Huen1969>

THE UNIVERSITY OF ALBERTA

A GRAPHICAL DISPLAY SUBROUTINE PACKAGE

by

© Wing Hing Huen

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE
OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTING SCIENCE

EDMONTON, ALBERTA

FALL, 1969

UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies for acceptance, a thesis entitled A GRAPHICAL DISPLAY SUB-ROUTINE PACKAGE submitted by Wing Hing Huen in partial fulfillment of the requirements for the degree of Master of Science.

ABSTRACT

An investigation has been made into the problem of providing the FORTRAN programmer with facilities that enable him to develop special-purpose, on-line, graphical applications programs. Two existing subroutine packages have been examined: the IBM 1130/2250 Graphical Subroutine Package and the VISTA Package for the Control Data 3600/DD250. This resulted in the development of GRIDSUB (GRID SUBroutine Package) to be used for on-line graphical display with a CDC GRID (Graphical Remote Interactive Display) that has been interfaced to an IBM 360/67. GRIDSUB gives the FORTRAN program, resident in S/360, an easy means of communication with GRID. With these routines, discrete portions of a picture may be displayed repetitively, modified, relocated or erased from the screen. GRIDSUB also allows the GRID operator to communicate with the applications program through manual input devices.

ACKNOWLEDGEMENTS

I am indebted to Professor J.P. Penny for his guidance and constructive criticism throughout the course of this research. Special thanks are due to Frank Jacobsen and Kenneth May whose helpful suggestions at several critical times gave GRIDSUB much of its present character. I am grateful to Brian V. Johnson and Linda Easton who have tested GRIDSUB extensively and provided examples of applications of GRIDSUB for this thesis.

I wish to thank the Department of Computing Science for the financial assistance and facilities which have made this project possible.

TABLE OF CONTENTS

	<u>Page</u>
CHAPTER I - INTRODUCTION	1
CHAPTER II - THE MAN-MACHINE GRAPHICAL COMMUNICATION	
2.0 Introduction	5
2.1 The Man-Machine Interactive Environment	5
2.2 The Display Console	9
2.3 Applications	17
2.3.1 A Computer Aided Logical Circuit Design System	20
2.3.2 Campus Planning	23
CHAPTER III - GENERAL-PURPOSE COMPUTER GRAPHICS SOFTWARE DESIGN	
3.0 Introduction	26
3.1 Software Support for a Console Command Language	26
3.2 Essential Features of a Graphical Subroutine Package	30
CHAPTER IV - A CRITICAL LOOK AT TWO GRAPHICAL SUBROUTINE PACKAGES	
4.0 Introduction	33
4.1 The VISTA Basic Subroutine (VBS) Package	33
4.1.1 Graphic Element Generation Routines	34
4.1.2 Block Handling	35
4.1.3 Modification of Blocks	39
4.1.4 Secondary Storage of Graphical Data	40
4.1.5 Manual Input Control	41

CHAPTER IV

Page

4.2	IBM 1130/2250 Graphic Subroutine Package	42
4.2.1	Initialization Routines	42
4.2.2	Graphic Element Generation Routines	44
4.2.3	Entity Handling	45
4.2.4	Updating Facilities	49
4.2.5	Attention-handling Subroutines	50

CHAPTER V - HARDWARE/SOFTWARE CONFIGURATION OF A GRAPHICAL DISPLAY SYSTEM

5.0	Introduction	53
5.1	Display Hardware	53
5.2	Data Transmission Between the Graphical Subsystem and the Main Computer	57
5.3	Operating System on S/360	62
5.4	Influences on the Design of the Software	67

CHAPTER VI - GRIDSUB, A GRAPHICAL SUBROUTINE PACKAGE

6.0	Introduction	71
6.1	Specification of GRIDSUB	71
6.1.1	Arguments Used by many of the Subroutines	76
6.1.2	Block Definition	78
6.1.3	Graphic Element Generation	79
6.1.4	Block Display	85
6.1.5	Modification of Display	86
6.1.6	Transmission and Decoding Routines	90
6.2	Discussion on Some of the Features of the Package Compared with the VISTA and IBM Subroutines	93

	<u>Page</u>
CHAPTER VI	
6.2.1 DLINE	93
6.2.2 BLOCK and ENDBLK Routines	95
6.2.3 TRANMT	96
6.2.4 Deletion Routines	97
6.3 Evaluation of GRIDSUB	97
CHAPTER VII - IMPLEMENTATION OF GRIDSUB	
7.0 Introduction	102
7.1 The BLOCK FILE and the DISPLAY FILE	105
7.2 Table Handling	109
7.3 Problem of Relocation in Displaying Picture Components	116
7.4 Recognition of Blocks in Light Pen Picks	121
7.5 Erasure of Blocks and Nested Blocks	126
7.6 Garbage Collection	129
7.7 The TRANMT Subroutine	132
CHAPTER VIII - HARDWARE WEAKNESSES IN GRID AND SUGGESTIONS FOR IMPROVEMENT	
8.0 Introduction	136
8.1 The Labelling Command, IDY	136
8.2 Return Jump Commands	142
8.3 Positioning Words	143
8.4 The Light Pen	144
CHAPTER IX - SUGGESTIONS FOR VERSION II OF GRIDSUB	
9.0 Introduction	146
9.1 Possible Major Improvements	147
9.1.1 Recoding the Package	147
9.1.2 Error Indication	148

	<u>Page</u>
CHAPTER IX	
9.1.3 Secondary Storage Facilities	149
9.1.4 Choice of GRID Supervisor	151
9.2 Additions to the Package	151
CHAPTER X - CONCLUSION	155
REFERENCES	159
APPENDIX A - GRID REPERTOIRE OF COMMANDS	165
APPENDIX B - FORMAT OF MSGTBL AND MESSAG	173
APPENDIX C - AN APPLICATIONS PROGRAM USING GRIDSUB	178
APPENDIX D - AN OUTLINE OF AN APPLICATIONS PROGRAM USING MESSAGE DECODING ROUTINES	183

LIST OF FIGURES

	<u>Page</u>
Figure 2-1	8
2-2	8
2-3	15
2-4	21
3-1	28
3-2	28
4-1	37
4-2	39
4-3	43
5-1	55
5-2	59
5-3	63
6-1	73
6-2	83
6-3	84
6-4	99
6-5	100
7-1	106
7-2	107
7-3	108
7-4	111
7-5	112
7-6	114
7-7	115
7-8	117
7-9	118
7-10	119
7-11	123
7-12	125
7-13	125
7-14	130

	<u>Page</u>
Figure 7-15	131
8-1	138
8-2	137
8-3	141
C-1	182

CHAPTER I

INTRODUCTION

In the past, man has been writing letters to, rather than conferring with, computers. The interaction between man and machine has been slowed down by the requirement that all communications be reduced to written statements. Moreover, in communications that involve strictly graphical data such as description of an architectural design or the construction of an electronic circuit, description in written statements has been found cumbersome.

Sutherland (Sutherland 1963) initiated the design of console command languages with which a console user can display graphic elements which are points, lines, arcs on the screen by using manual input devices which are the keyboard and the light pen. The console user can impose constraints on the graphic elements such as setting lines parallel or perpendicular to each other. Any configuration that has been defined may be replicated any finite number of times. Graphic elements can be assembled together to form picture components and picture components can be further assembled and so on to form a structure that can be dealt with, moved on the screen and modified.

The console command language which is the only interface visible to the console user has to depend on the software support of a package of graphical subroutines and

other lower level system programs. The graphical sub-routines may be called with a high level procedure language such as FORTRAN so that the applications programmer can design and implement this console command language without involving himself with the hardware and low-level system programs.

This thesis is concerned with the design and implementation of a graphical subroutine package, GRIDSUB. Since the package acts as a medium between the low-level system programs and the applications program which interprets the console command language, requirements for console command languages and the low-level system programs are fully considered.

Two similar subroutine packages, the VISTA (Abraham 1966) and the IBM 1130/2250 packages, one from a research organization and the other from a computer manufacturer, are evaluated from the point of view of an applications programmer. These two earlier efforts served as a guide for the design of GRIDSUB.

GRIDSUB is to be used by applications programs which are in an IBM 360 model 67 to provide interactive display on a CDC Graphical Remote Interactive Display (GRID) Subsystem (CDC 1968). The GRID display unit is attached to the S/360 with a telecommunication line. The influences that the hardware/software system of S/360 and the mode of transmission have had on the design of the GRIDSUB

are also discussed.

The package, GRIDSUB is presented in Chapter 6. It contains graphic element generation routines, which generate line segments, points and characters; facilities for structuring blocks of graphical data, nesting of blocks; routines for modifying the displayed picture; and routines for controlling communication between S/360 and GRID.

Blocks of graphical data corresponding to picture components are converted into GRID subroutines which are kept in a Display File. The locations of the GRID subroutines are recorded in a Display File Table. To display the picture components, GRID calling sequences to the GRID subroutines are inserted into the Display File which is then transmitted to GRID for execution. When the GRID display is interrupted by manual input devices, the GRID subroutine in which interrupt occurs is found by comparing interrupt addresses with the Display File Table. The management of files and tables for the construction and modification of the Display File is described in detail.

Some weaknesses in the design of the GRID machine have been found during the implementation of the package. Special programming efforts have had to be performed at the expense of length of program to avoid these idiosyncrasies. Suggestions for improvements on the design of

hardware are made.

The first version of GRIDSUB is ready for release to the user public. However, the package is an evolving system, and some extensions to the package are being implemented. A second, more efficient version of code with a view to saving S/360 core space and equipped with secondary storage properties is being planned.

CHAPTER II

THE MAN-MACHINE GRAPHICAL COMMUNICATION

2.0 Introduction

This chapter explores the use of a display console as a link between a man and a computer. This man-machine communication is available both to the programmer well versed with computer technology and, where special-purpose systems are available, to the console user who has no detailed knowledge of programming.

The equipment generally available on a general purpose display console is discussed. A brief description of the cathode ray tube, the display processor, the line and character generators, the light pen and the keyboard is included.

To demonstrate the important role of on-line computer graphics and its far reaching implications, some of the graphical applications that have already been in service are presented. Two applications of computer aided design with graphics consoles are reviewed in detail.

2.1 The Man-Machine Interactive Environment

An often quoted sentence of Richard W. Hamming (Hamming 1962) states that 'The purpose of computing is insight, not numbers'. Usually in a computer output of numbers, the most important part is the inter-relationship among the numbers. However, traditionally the major role

of the computer is seen as the calculation and printing of an answer. A numerical answer is useful only when it is understandable to human beings. All too often a computer programmer, in dealing with large amounts of numerical data, loses sight of what he is doing. Diagrams, drawings and graphs are essential tools for human understanding in many scientific fields. Working in batch-processing mode, a computer programmer is often forced to code his drawing into numerical data for the computer and convert the computer-printed numerical output into drawings or graphs. This process is time-consuming, preventing him from trying many examples. If a man can specify his problem in his own terms and receive his results in a readily understood form, the computer can be a more useful tool to him. Thus the use of graphical input/output devices provides more meaning than simple streams of numbers.

In a typical batch processing environment a computer user may have to wait for several hours before he can get the output. All too often he may find that the computer has fruitlessly laboured to satisfy constraints which are obvious to a human being to be mutually incompatible. Human intervention would have been desirable in the course of execution of his program. However, because of man's slow response time he has had to be isolated from the computer operating in microseconds.

Sutherland in his SKETCHPAD graphical program (Sutherland 1963) first demonstrated the possibility of man-machine communication through line drawings on the graphical display console. SKETCHPAD and the advent of on-line systems open a new era in which man and machine can be joined in an intimate co-operation, a combination that uses the creative, imaginative power of the man and the analytical and computational powers of the machine.

Time-sharing allows one computer to service many users concurrently. The mismatch between the response times of man and computer is no longer an economic barrier to letting a user interact directly with the computer. Many problems, which previously could not be solved on the one hand by man alone because of the large amounts of data to be processed, or on the other hand by machine alone because of the requirements for human judgment and insight in frequent redirection of the solution algorithm, may now be solved for a man-machine team. This conversational mode is widely believed to be capable of stimulating creative human thought (Allen et al. 1964), (Licklider et al. 1962), (Coons 1963). Moreover, any wrong decision on the part of man does not affect the overall operation. Exploration of paths by trial-and-error methods become more practicable.

The way in which the graphical display is used is simple and straightforward. User-computer communication is depicted in Fig. 2-1.

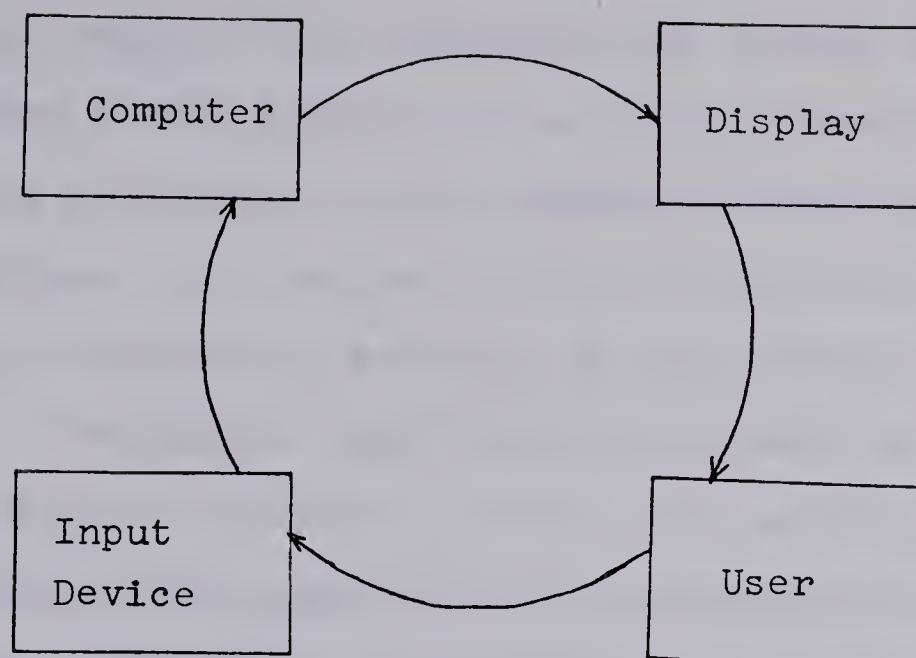


Fig 2-1 The man-computer-display loop.

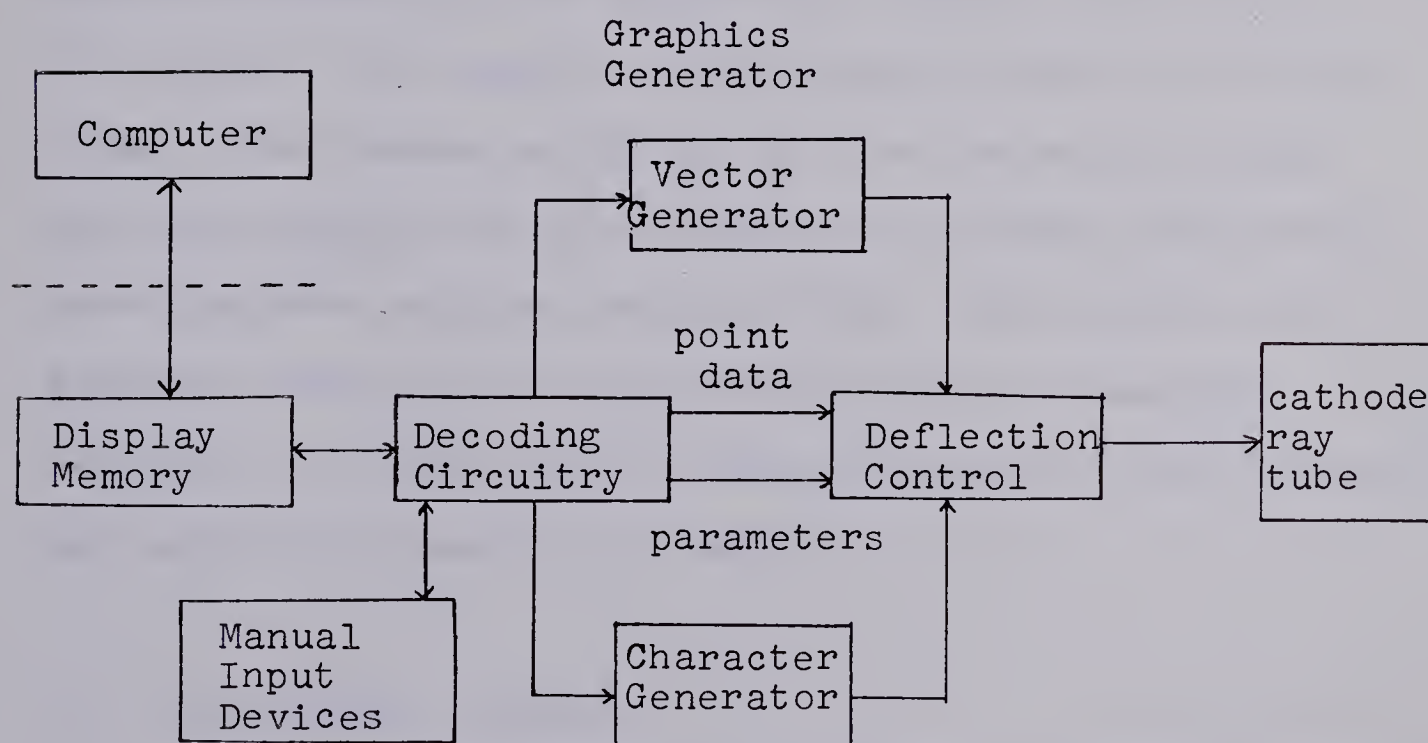


Fig 2-2 Block diagram of a typical display console.

First, the console user formulates his design or presents his problem to the computer in a console command language, specifying his action to the computer through manual input devices. All he has to do is to specify a logical sequence of operations governed by the console command language. The manual input devices activate an applications program which is invisible to him. He need not be concerned with the operation of the applications program, which is developed by an applications programmer who is a computer specialist in the field. The applications program tests the hypothetical design or analyzes the problem. The output is also in graphical form. If the output is satisfactory, the problem is solved. Otherwise the user modifies his original concept by changing certain parameters and repeats the process until he is satisfied. The communication is exclusively in pictorial terms. The concept which may at first have been vague and crude evolves in the interactive process and becomes more concrete and precise at the end. New concepts are generated through the repetitive process of computer analysis of a crude model, human evaluation of the analysis and decision to modify the model.

2.2 The Display Console

The display console is the active interface between man and machine. It serves mainly in converting sequences

of computer generated digital commands into a picture for viewing by the console user and it includes manual input devices for entering console command language statements. The display console in general can be represented by Fig. 2-2.

The computer generates display commands and stores them in the memory storage of the display device. These commands are repetitively fetched from the storage and converted by the decoding circuitry into analog drive signals. The graphical generator, triggered by the drive signals, provides beam deflection control of the cathode ray tube (CRT). The controlled motion of the electron beam forms the image on the screen.

The CRT resembles a television monitor in external appearance. Its operation differs greatly in the motion of its electron beam which can be randomly positioned. The screen can usually be considered as a grid of 1024×1024 co-ordinate raster to which points and characters can be positioned. Line segments can be drawn between any two raster points. The inside surface of the screen is covered with phosphorescent material which usually glows for a short time in the order of microseconds after being energised. If the commands are not repetitively fetched from the storage, the image will fade. If the repetition rate is not frequent enough, a flicker will result. Typical repetition rates are from 20 to 60 times a second.

Some display devices do not possess dedicated storage. The display commands have to be repetitively supplied from the central computer to produce a continuous display. In the display devices with memory storage, the display commands need be generated only once by the computer or when a change is made, thus releasing the central computer for other tasks. If the memory is cyclic in nature, for example, drum or disc, picture refreshing is simple. The repetition rate of the picture is the cycling rate of the memory. If the memory is non-cyclic like core memory, transfer instructions are necessary, and a timer instruction is also needed to control the repetition rate. Magnetic core storage is preferred to drums or discs as modification of the display program is simpler. The transfer instruction in display devices with core storage can also provide an important feature, sub-routining. A displayed picture very often consists of certain basic geometrical shapes, such as a transistor in an electronic circuit diagram, which are often repeated. Subroutining can greatly reduce the required storage, as only the display commands for one copy of the object are required in the storage and, whenever needed, a calling sequence supplying the starting co-ordinates for each copy transfers control to the storage location for the basic representation of the object.

The response to user requests of on-line graphical

devices is expected to be within or very close to human reaction time. The turn-around time includes the two-way transmission time, time elapsed in gaining access to the central computer whose resources are being shared, and computing time to service the request. Many devices, (Ninke 1965) (Prince 1966), incorporate local computing power to reduce the turn-around time. These devices can be represented by Fig. 2-2 with the block for decoding circuitry replaced by a small computer. Besides being a display processor, the small computer may perform simple manipulations such as handling of manual input attentions and rearrangement of display commands, while complex processing that requires a large data base and considerable computation is handled by the central computer. Sometimes this remote computer can be a quite big computer such as the IBM 1130. This can be uneconomical because the CRT and the remote computer constitute a terminal being monopolised by a single user.

Characters can be formed in a number of ways. The most important ones are the point positioning (eg. DEC 340), the stroke positioning (IBM 2250) and the CRT beam extrusion control method (eg. Stromberg-Datagraphic SD 1090 Direct View Display). A special character generator is usually used in the first two methods to convert a character code into a sequence of dots or short strokes. The last method mentioned forces the electron beam through a mask which

causes the beam to take the shape of the character. The last method is fast but difficult and expensive to make alterations on character formation. Many display consoles provide several sizes of characters, vertical and horizontal orientations, subscripting and superscripting.

Lines are typically generated by one of two methods: approximating the line by positioning a series of points on raster positions, or creating a continuous solid line between raster positions. The display commands need only specify the starting and terminating co-ordinates (absolute mode) of the line segment or a starting point and the X and Y increments (relative mode).

The manual input devices include the light pen, the alphanumeric keyboard and function switches. The user may point with the light pen at the blank screen to indicate a position or at displayed picture components. The user may also enter parameters and alphanumeric data through a keyboard and specify desired functions with pushbuttons or switches.

The light pen (Allen et al. 1964) which is a light sensing device, consists of a hand piece with a mechanical shutter at one end of a fibre optics bundle. The other end of the bundle provides input to a photomultiplier system. To prevent random detection from sources other than the screen, the photomultiplier is usually only sensitive to strong intensity light of a particular color.

A signal is produced only when the shutter on the pen is open and an image is displayed in the field of view of the pen. Henceforth the term attention is used to indicate the signal generated when the console user depresses a function key or an alphanumeric keyboard key, or points at a displayed picture component with the light pen and opens the light pen shutter. The attention will stop the display cycle and the display processor then examines the status of the display such as the address of the display command where the execution is stopped and contents of the X,Y deflection registers, which indicate the position of the light pen.

Many display processors have labelling commands which place labels (also called identification numbers or correlation values) in a special register. The labelling commands are non-displayable and are inserted among other display commands. The label can be obtained readily by the display processor for interpretation after display attention.

Since the light pen is a sensing device, it cannot be used directly to enter positional information on part of the screen where no image is displayed. This problem is often solved by the 'tracking' method (Gurly et al. 1959, Stotz 1963). A common tracking technique is the 'centroid' method. A cross is formed by sequentially displaying series of points a,b,c,d from the outside to

the centre (X_0, Y_0) of the cross as shown in Fig. 2-3.

As the series a of points is generated and the light pen senses light, the display stops and the co-ordinate value Y_1 is read from the Y register. The generation of series a is abandoned and that of series b is started. Assume the Y co-ordinate value obtained is Y_2 . In the next two stages, 2 positions X_1, X_2 from lines c and d are read from the X deflection register. The cross can then be generated by the tracking program with the centre of cross at a new position (X_n, Y_n) where

$$X_n = \frac{X_1 + X_2}{2}$$

and

$$Y_n = \frac{Y_1 + Y_2}{2}.$$

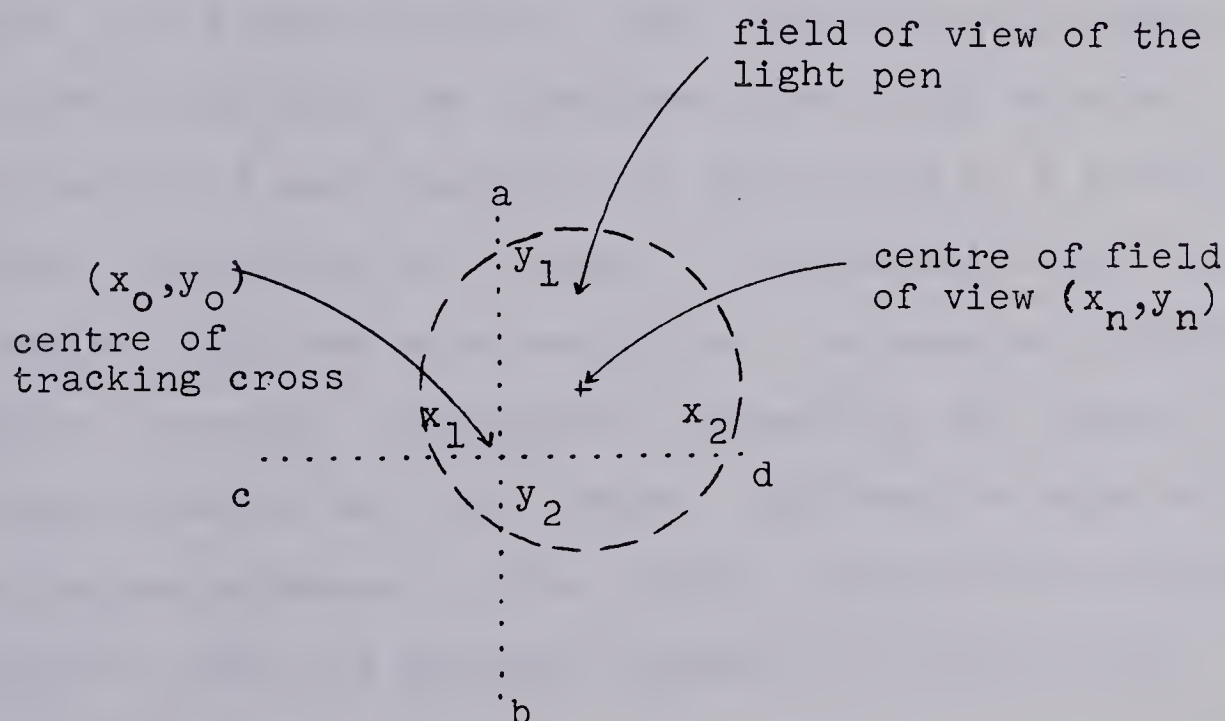


Fig 2-3 The tracking cross.

Simpler methods for positioning can be used such as the scanning method. The console user points the light pen at a desired position and a moving matrix of points or a set of horizontal and vertical lines is displayed on the screen until a light pen attention is obtained. However, the 'centroid' method of tracking allows more accurate following of the path of the pen. This feature is necessary if there is to be line drawing with the light pen.

Alphanumeric input to the display can be entered through keyboard devices similar to a typewriter. Display units solely devoted for displaying textual material -- alphanumeric characters, have hardware facilities such that pressing a key will cause the code for that character to be placed in the memory and the same character will appear on the CRT screen with the characters previously entered. Positioning of each character is determined by a cursor. Hardware facilities for erasure of individual characters or entire lines are also available. For general purpose graphics consoles, alphanumeric characters are usually not entered directly into the memory. Software is required in the display processor or the central computer to interpret character codes and generate commands to display each character entered.

Switches on the keyboard can also be programmed to have specific functions. Function switches are usually used in combination with the light pen and alphanumeric

keyboard to indicate the desired action to the computer applications program. For example an operator can press a function switch designated by the applications program to mean ERASE, and point to an element with the light pen to indicate that the element is to be erased from the screen.

2.3 Applications

Applications of the graphical display have been implemented in many areas including engineering drawing (Chasen-1 1965), animated movies (Knowlton 1964), graphical information retrieval (Morse 1968), data reduction and analysis (Britt et al.), computer aided design (Jacks 1964, Dertouzos-1 1967, Dertouzos-2 1967, Cross 1967), perspective drawings (Romney 1968). Some of the uses are briefly discussed here and a closer look is taken at two examples of computer aided design.

A large number of drawings is required to describe a complex item, each showing only a small aspect of the object. The price of keeping these drawings up to date is high because changes in one part of the design necessitates re-drafting of many drawings. With a digital computer, a highly accurate numerical model of the entire drawing can be kept in a mass-storage unit. If one designer modifies the data representation of one component of the object, all other occurrences of the component can

be also modified automatically. He can even prepare digital tapes from the stored data for numerically controlled cutting tools (Chasen-2 1965).

Moving pictures on the display are valuable in areas such as the operation of some mechanical devices or in illustrating relative movement of objects in some system of objects. For example, a satellite's motion can be described by complicated differential equations for which numerical integration by computer is required. Zajac (Zajac) determined by iterative procedures the position, velocity, orientation and angular momentum of the satellite for successive moments of time and showed the results of computation as a series of drawings which formed a movie. The satellite is drawn in perspective, both from a vantage point in space and from a vantage point travelling in orbit with the satellite.

The display can be used as a graphical information retrieval system with capabilities for modification and addition to the stored information. A drawing stored in the computer can be regarded as being drawn on a large imaginary surface. The display can be considered as a view box (or a zoom lens) which can be moved to examine an area of interest or "zoomed" in or out to show parts of the drawing in greater or lesser detail. A display device has been used in a meteorological system (Wallington 1968) to display a map to which the operator can supply data from

weather stations; the computer can then calculate and display isobars and rainfall patterns, say, on the original map. In map retrieval systems (Morse 1968), the operator may add boundaries or shade regions. Contour maps might be decomposed or several maps merged to form one.

Data reduction and analysis typically require many iterations and considerable processing of data in the solution of the problem. Normally several hours of turn-around time are required in a normal batch system. But often the analyst may find that the whole set of data was bad, and required corrections or should be discarded. With the help of on-line graphical terminal, many iterations and decisions can be made in one session. In a typical analysis system, digital data can be read by the computer, scaled and displayed in the form of curves. The console user can close gaps or eliminate obviously erroneous points by adjusting data points with the light pen and function keys. A recent example is the on-line statistical computation on data obtained from medical research (Britt et al.)

The graphical display has also been used to produce perspective drawings of air frames (Prince 1966), (Chasen-1 1965), motor cars and buildings as well as three dimensional representations of geometrical objects (Romney 1968). It is possible to display from a point of view and

with a scale specified by the console user.

Perhaps the most important applications for graphics are in the area of computer aided design. The display device is widely used in the design of electronic circuits (Kuo et al. 1969, Cross 1967) as well as architectural design (Newman 1966), and urban planning (Horwood 1968). Two applications which are under development in the University of Alberta are described in detail in the following paragraphs.

2.3.1 A Computer Aided Logical Circuit Design System

A logical circuit is a set of interconnected basic logic devices, i.e. AND, OR, and NOT gates, to perform some complex logical function. The inputs and outputs of each of these devices have only two discrete values so that the principles of mathematical logic can be applied.

An applications program, CALD for Computer Aided Logical Design (Johnson 1969), is being developed so that a console designer can construct and modify his circuit through manual input devices. As far as he is concerned, the CRT screen is divided into two areas: the control area on the left and the display area. Basic components of the circuit and a set of command words are displayed in the control area as shown in Fig. 2-4.

The console designer can indicate his desired operation by pointing the light pen at the command words

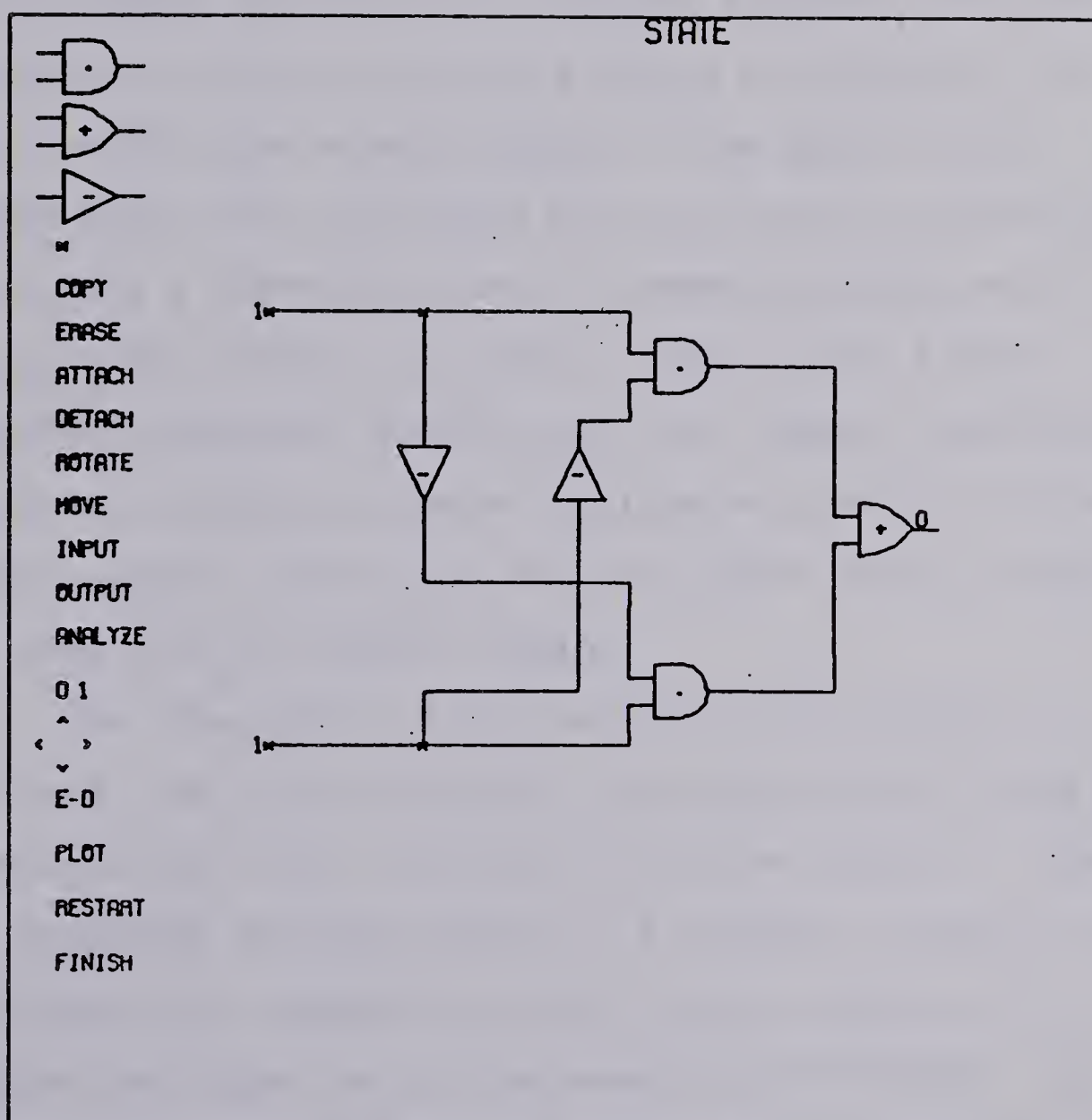


Fig. 2-4 A CALD display. (Courtesy of B.V.Johnson)

and the logical components or by depressing the keyboard. His actions are governed by a console command language which is specified and processed by the applications program CALD. The applications program aims at providing a set of possible operations which are special to this application but are most natural to the console user. For example, if he wishes to display an AND gate at position P in the display area, he points the light pen at the command word COPY, the logical component AND gate and indicates with the scanning method the position P at which the AND gate should appear. If he points at the command words COPY and DETACH the two light pen attentions do not form a legitimate console command language statement and CALD displays an error message on the screen. The console designer can thus copy the logical components, join them to build up a more complicated circuit with the ATTACH command, delete them with the ERASE and/or disconnect them with the DETACH command.

When the initial construction of the circuit is completed, the console designer can supply input values, represented by 0 and 1 in Fig. 2-4 to the circuit. Since this automated "drawing board" is a computer terminal, he can request the computer to check out the circuit by pointing the light pen at the command word EXECUTE. All the commands are given with the light pen or by pressing a key on the alphanumeric keyboard or function keyboard.

He can use the system with no knowledge of details of the CALD applications program which interprets all his actions.

2.3.2 Campus Planning

The primary objective in good campus planning (Hough 1968) is to make the best use of the site and its resources and to ensure that complex activities of the campus will complement each other. The general planning includes the analysis of topography, considerations on community activities within the campus, services like utility lines and drainage patterns, vehicular traffic, and pedestrian movement patterns.

To solve the planning problem, a map of the campus has to be stored in the computer. With this map displayed on the screen, the console designer can develop with input devices resource maps showing existing services and maps showing plan of buildings to be erected. These maps can be overlayed to obtain a map of common regions. Factors such as usage of the buildings, accessibility and necessity for modification of utility lines can now be studied. Decisions pertaining to the campus development can be made from the resulting maps.

Several problems must be solved to make the application feasible. They include:

1. Line-drawing capabilities on the screen with the light pen so that the console user can develop plans with

free-hand drawing.

2. Selection of a suitable encoding method to store the map of the campus.
3. Development of efficient algorithms to determine the intersection of regions, their area and perimeter; and methods for shading the regions.

The tracking method can be modified to provide line drawing properties. In the tracking mode the tracking cross follows the motion of the light pen but only the last position of the cross is of interest. For line drawing all the previous positions of the centre of the cross are stored in the memory and are regarded as end points of line segments. The path of the light pen is shown on the screen with a plot vector command.

Several encoding methods are commonly used: the X,Y co-ordinate method (Mezei 1968), chain encoding method (Freeman 1961-1, 1961-2) and the skeleton encoding method (Pfaltz et al. 1967, Rosenfeld et al. 1966).

No one method is best in all situations (Deecker 1969). The XY co-ordinate method and the chain encoding method are good in coding open curves, determining area and perimeter of regions, and in building the display file. The skeleton method is best in determining the intersection of regions, rotation of figures and shading of regions. A map drawn on the display may be converted for XY co-ordinate encoding with a minimum of computing time. The presently

planned implementation involves only simple outlines of regions which are polygons of less than ten sides. The use of the XY co-ordinate method should result in the most economical use of storage. Moreover with the restriction of not more than four points of intersection, the XY co-ordinate method is usable in determining the intersection of such regions. The XY co-ordinate method has been chosen for this application.

CHAPTER III

GENERAL-PURPOSE COMPUTER GRAPHICS SOFTWARE DESIGN

3.0 Introduction

The concept of a central complex information structure which contains associations among the parts of a displayed picture is discussed. Three levels of software support for a console command language are considered: the special-purpose applications program, the library of general-purpose graphical subroutines and hardware oriented system programs. Emphasis is laid on the essential features of the mid-level graphical subroutine package which is the main concern of this project.

3.1 Software Support for a Console Command Language

The central requirement for interactive operation through use of a display console is a computer-stored data structure which contains not only the data attached to a picture but also the associations and interrelations among the picture components. As an example in the logical circuit design, the interconnection among the logical components as well as their types and input and output values must be stored in a data structure. Each console command activates the applications program which may construct the data structure initially, handle and modify it. The relation of the console user and the applications

program to the central data structure is shown in Fig 3-1. When the picture components are entered and structured, and parameters and constraints attached, the data structure contains a complete representation of the drawing on the screen. The applications program can now do processing using the data structure to generate further information concerning the problem, such as computing the output value of a logical circuit from its input values. To give the console user graphical feedback, the applications program updates the display file and produces a pictorial representation of the contents and structure of the data.

The console command language is the final interface between man and machine at the console. Since this language is most natural to the user, it must be oriented to a specific problem. Thus many console command languages may exist, each for a particular problem area. Examples of these high-level, user-oriented language systems are MIT SKETCHPAD (Sutherland 1963), SKETCHPAD III (Johnson 1963), Bell Animation Language (Knowlton 1964), SDC GPDS (A.H. Vorhaus 1966), GINA (F.K. Kuo et al. 1969). However, these console command languages have common requirements imposed on the applications programs, the most important of which are facilities for handling the central data structure, constructing and modifying the display file and handling attentions from console input devices. If a package of routines is developed to provide the applications

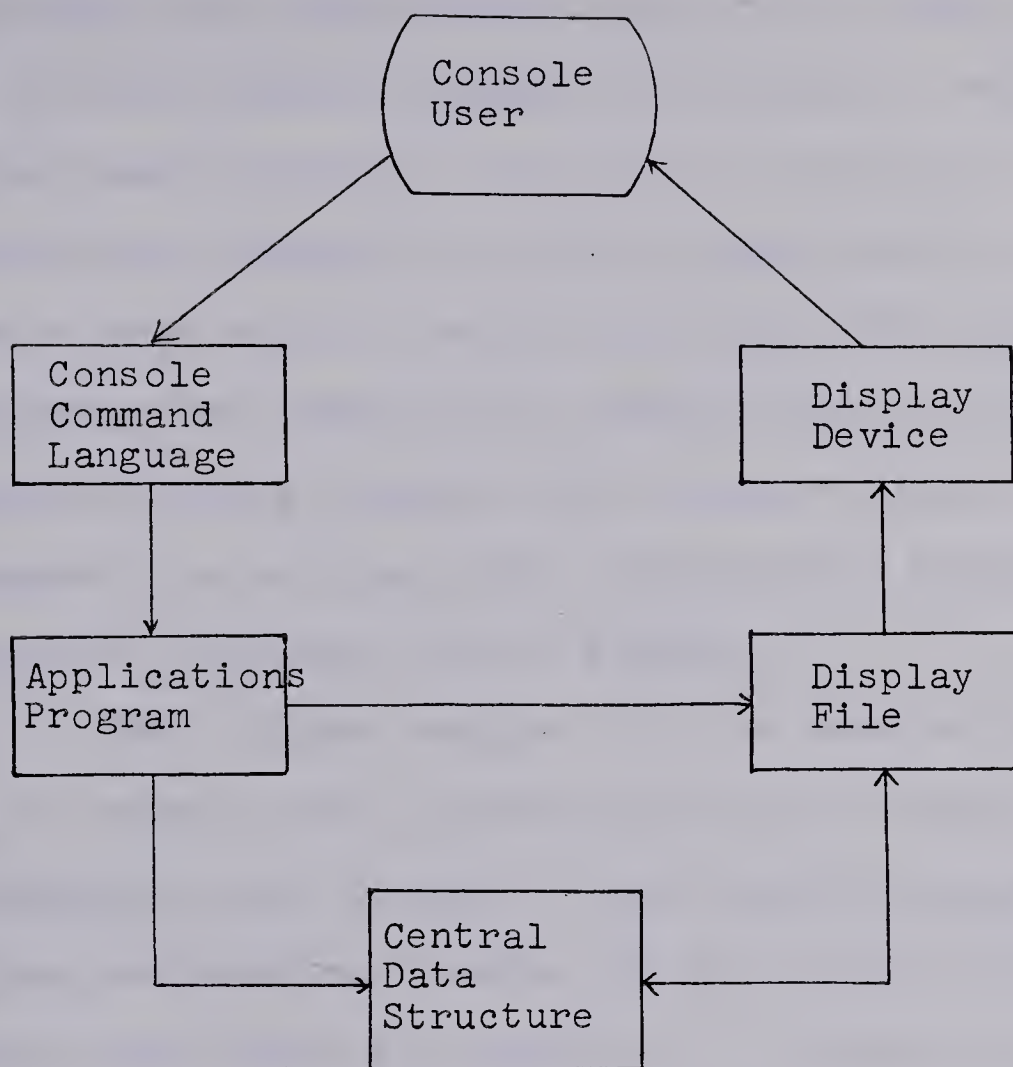


Fig 3-1 The central role of the data structure.

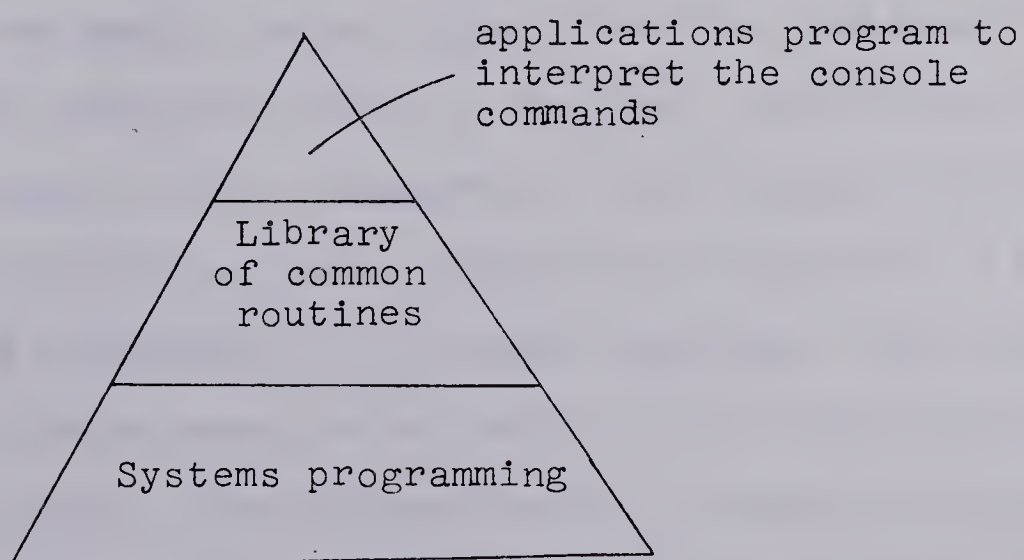


Fig 3-2 The foundation for console command languages.

programmer with these common facilities, development of a new console command language will be easy. Requirements of the common routines are listed in section 3.2. These routines are designed to save the applications programmer time or work which is mainly clerical. The applications programmer can construct the display file, control data transmission and organize the necessary display core memory management by calling these routines with parameters in a high-level language such as FORTRAN.

The software support for the console command language can be shown by the pyramid structure in Fig 3-2. At the foundation of the pyramid is the system programming support. System programming provides the most basic manipulating capabilities which are essential in forming the higher level package of common routines.

The system programs include the operating system of the main computer, transmission routines and hardware oriented routines that handle manual input attention and tracking.

The operating system in the main computer performs functions such as file management, input/output supervision, storage allocation and task scheduling to maintain a multi-programming environment in the main computer. The graphical task competes among other tasks for the services of the operating system. The disposition of a request for services (eg. the order in which requests are handled relative to other requests, priority rules etc.) is dictated by the

queueing and dispatching algorithms of the operating system. On-line tasks such as graphics usually have the highest priority.

Transmission routines may be channel programs that govern the flow of graphical data between the computer and the display unit or special routines to transmit or receive graphical data along telecommunication lines. Transmission routines must be implemented at the lowest level because they **must** conform to the specification of the hardware interface.

Programs are also required, either resident in the main computer or the display processor to process attentions to the display device. These attention handling routines, loosely called the display supervisor, have to identify the cause of the attention, read associated device registers, convert the codes which are generated by attentions from the manual input devices and arrange the converted codes so that they can further be processed.

3.2 Essential Features of a Graphical Subroutine Package

The console command language is directed to the user who is not necessarily a programmer. The implementation of the console command language makes use of the graphical subroutine package which forms the mid-level software support. The package is programmer-oriented.

The following functions are considered to be desirable

in a graphical subroutine package:

(i) Graphic element generation:

- a. Given an (X,Y) co-ordinate pair, place the CRT beam at the point on the screen.
- b. Draw a line segment or a series of line segments.
- c. Create a character or a string of characters.
- d. Give names and allocate storage for graphical data representing pictorial components.
- e. Display a picture component repetitively at a number of positions on the screen but defining the picture component only once.

(ii) Transmission control and interpretation of graphical data:

- a. Call transmission routines to pass graphical data to the display.
- b. Accept and interpret the attention information from the graphical device.
- c. Uniquely determine a picture component from attention information when the light pen is used.
- d. Recover co-ordinate position of any picture component.

(iii) Display core management.

- a. Blank out a picture component without losing the graphical information.
- b. Unblank an inactive picture component.
- c. Relocate a picture component on the screen.
- d. Destroy the graphical data that is not needed in the display core.
- e. Rearrange by garbage collection routines the commands in the display core, to retain as much free space as possible so that the display core is most efficiently used.
- f. Store graphical data onto direct access files so that it can be used in future sessions.

CHAPTER IV

A CRITICAL LOOK AT TWO GRAPHICAL SUBROUTINE PACKAGES

4.0 Introduction

Two graphical subroutine packages are scrutinized in this chapter in view of the requirements suggested in Chapter 3. This chapter does not attempt to describe the packages but tries to evaluate them from a user's point of view in terms of programming convenience and the facilities provided. Areas such as graphic element generation, management of picture components, updating facilities and control of manual input devices are covered. Only the more important subroutines and their arguments are described to illustrate particular features of the packages.

4.1 The VISTA Basic Subroutine (VBS) Package

The VISTA Basic Subroutine Package (Abraham 1966) is a set of FORTRAN subroutines developed in the Computer Research Section of the Commonwealth Scientific and Industrial Research Organization for use on a CDC 3600 to which the VISTA DD250 visual display unit is connected via a channel. The package contains subroutines for graphic element generation, construction and modification of units of graphical data, secondary storage and control of manual input devices.

4.1.1 Graphic Element Generation Routines.

The basic graphic elements in a picture are points, line segments and characters. There are seven routines for generating graphic elements:

<u>Subroutine name</u>	<u>Function</u>
MOVE(X,Y)	moves the beam from the present position to an absolute position (X,Y) of the screen.
MOVER(X,Y)	moves the beam by increments given by X and Y
POINT(X,Y)	displays a point at absolute screen position (X,Y)
POINTR(X,Y)	displays a point at (X_i+X, Y_i+Y) if the initial position of beam is at (X_i, Y_i)
VECTOR(X,Y)	draws a line segment from present beam position to absolute position (X,Y)
VECTORR(X,Y)	draws a line segment from initial beam position (X_i, Y_i) to (X_i+X, Y_i+Y)
VTEXT(-,-,-,-)	displays a string of characters with choice of 4 possible sizes, 2 intensities and 2 orientations

The point, vector, and line segment generation routines do not have initial beam positioning properties. The MOVE or MOVER subroutine has to be called often to set the beam position. Assume a string of characters is to be displayed with the first character at (200,300) while the current beam position is (100,100) the following calls are necessary:


```
CALL MOVE (200,300)
CALL VTEXT (-,-,-,-)
```

The line and point drawing routines display only a line segment or a point at a time. It is very inconvenient to display a series of points or to draw line segments through a number of points which are given in absolute co-ordinates. One call is necessary for each segment. The only way to avoid lengthy programming is to define each point as relative to the previous point. The programmer may then use VECTORR in a DO loop.

Assume POINTX and POINTY are two arrays of 100 elements. The (I+1)th element contains the X or Y increments of the (I+1)th point relative to the Ith point; where I is between 1 and 99. The contents of the first element is relative to the beam position. The line can be drawn with

```
DO 51 I = 1, 100
51 CALL VECTORR (POINTX(I), POINTY(I))
```

4.1.2 Block Handling

A picture for display can be divided into a number of discrete picture components such as a transistor in an electronic circuit design application. The graphical data for a picture component is assembled in units called blocks. A block is composed of commands which generate graphic elements i.e. points, line segments and strings of characters. A block is established by preceding calls to

graphic element generation routines with

```
CALL BLOCK (NAME) and ending with
```

```
CALL ENDBLK without arguments.
```

The argument NAME is an 8-character alphanumeric code. A well defined symbolic block name may serve as an aid in documenting programs and certainly improves readability. However, the use of alphanumeric characters instead of integers as block names eliminates the advantage of defining blocks within DO loops.

A completed block will not appear on the screen unless a request for display is issued by the following statement:

```
CALL DISPLAY (NAME, IX, IY)
```

where NAME is the block name.

The FORTRAN programmer has to be cautious in setting the screen position (IX,IY) at which the block is to be displayed. The position is not the origin of the block but the lower left hand corner of an imaginary rectangular frame circumscribing the block.

For example, consider a block representing a triangle ABC with one side 200 and altitude 300 raster units as shown in Fig 4-1. The following statements define the block and display the triangle with its centroid at the centre of the screen (511,511).

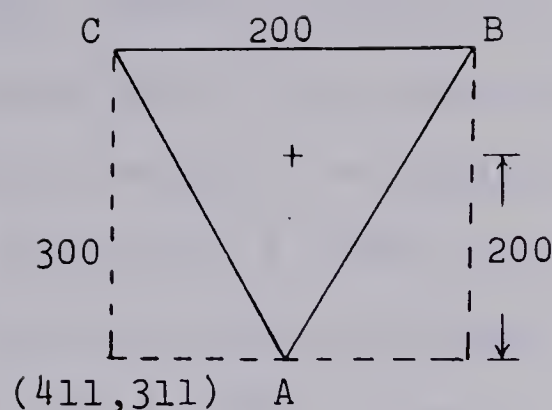


Fig 4-1.

```

CALL BLOCK (TRIANGLE)
CALL MOVER (100,0)
CALL VECTRR (100,300)
CALL VECTRR (-200,0)
CALL VECTRR (100,-300)
CALL ENDBLOCK
CALL DISPLAY (TRIANGLE,411,311)

```

The definition of the block cannot start from the point A. The CALL MOVER is necessary for setting the position of the beam. The CALL MOVER and some calculations could be saved if the origin of the block, the point where definition begins, were the same as the point at which the block is to be displayed.

Only one block can be defined at a time. A new block cannot be defined if the previous one is not completed. Moreover, nesting of blocks is not allowed i.e. a block cannot incorporate a previously defined block as one of its elements. Thus the package does not provide facilities for constructing interrelations among picture components.

A block under construction can also be displayed at (IX,IY) by the following statement:

CALL INSTANT (NAME,IX,IY).

This feature may be very desirable in applications like logical circuit design. An applications programmer may use a block containing a number of line segments to represent wires joining logical elements. He may provide the console user with a light button so that the latter, in the process of concatenating wire segments, may at any time leave the current wire incomplete and request for a display of the status of the design (P. Cross 1967). The INSTANT call displays the incomplete block without providing errors.

A block, once defined, can be displayed at a number of positions on the screen. The package has another defect in that the programmer cannot provide any labelling information to distinguish blocks with the same name and composition but displayed at different positions. The package must recognise identical blocks by means of their co-ordinates. But this method is valid only if their imaginary frames do not overlap. For example the wires ABCD and PQRS, Fig 4-2. have the same block definition but are displayed at A and P. Confusion would result in attempting to distinguish between the blocks by means of co-ordinates.

inserts a character which is stored in the uppermost 6 bits of BUF at position (NX,NY) of a completed block called NAME which is being displayed at position (IX,IY). The orientation of the character is in +X or +Y direction according to whether the 8th parameter is +1 or -1. The size, type (italic or normal), light pen detectability, blinking and intensity of the character are set by NS,I,LP,BLINK and INTENSITY. Since IX,IY are optional, IH\$ is used as a separator. Graphic elements in a defined block can also be deleted one at a time through the DELETE subroutines which have similar arguments as the INSERT subroutines.

As can be observed from the example, these routines are inconvenient to use. Moreover only a character, a line segment or a point can be inserted in each call. A routine which initiates the modification mode and followed by the usual graphic element generation routine calls would be more useful.

4.1.4 Secondary Storage of Graphical Data

Only forty blocks can be defined at a time and resident in core. To overcome the limitation, a programmer can store onto drums some of the blocks which are not needed immediately with the SAVEBLOCK subroutine, and subsequently destroy the contents and names of these blocks with the DESTROY routine. Blocks stored on drums can be

copied into core by the TAKBLOCK subroutine. These two subroutines are very useful in saving core space.

4.1.5 Manual Input Control

The VBS package provides the following routines to control the function keyboard and light pen:

(1) LIGHTPEN (BLOCKINT,TRACKING,IX,IY) initiates either the capture mode and the tracking mode of the light pen with the tracking cross displayed at (IX,IY) and the program waits for an attention from the light pen. If light pen attention occurs at one of the user's own blocks, control is transferred to a user defined routine BLOCKINT which modifies the display. When light pen attention occurs at the tracking cross, control is passed to the tracking routine TRACKING which has to be defined by the programmer. The status of the registers of the display at the time attention arises is stored in a common area.

(2) KEYBOARD (KEYINT) subroutine always accepts attentions produced by the keyboard and passes the control to the user-defined routine KEYINT which interprets the key code.

These routines are inefficient in that they simply recognize the cause of manual input but leave the programmer the responsibility of interpreting the keyboard codes or handling the status information. The interpretation of keyboard codes and conversion of status information into

more easily handled form should be provided by the subroutine package or lower level system programs.

Moreover, every manual input attention produces an interrupt in the CDC 3600 which is processing programs besides the graphics applications program. This mode of attention handling imposes considerable overhead in the execution of the 3600.

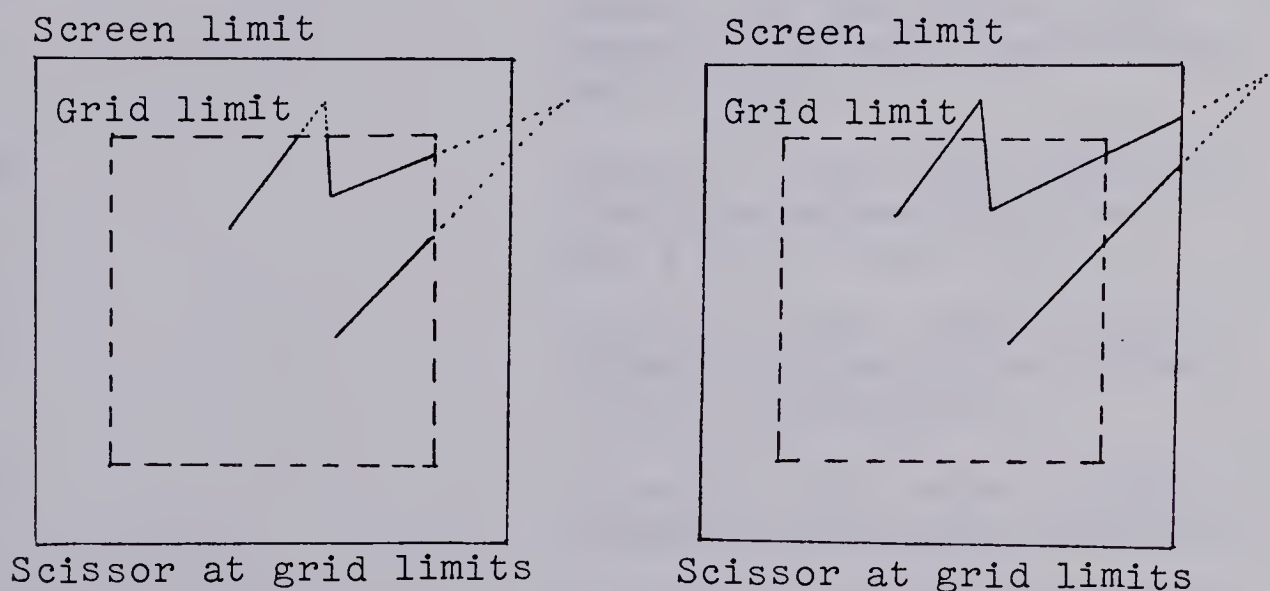
4.2 IBM 1130/2250 Graphic Subroutine Package (IBM Form C27-6934-0)

The Graphic Subroutine Package (GSP) allows a FORTRAN programmer to create displays on an IBM 2250 Display Unit Model 4 attached to an IBM 1130 computing system having at least 16K words (16 bits) of core storage and one disk. Displays are produced on the basis of control information and data supplied by the programmer in the call to each GSP subroutine. The control information and data define what is to be displayed and where it is to be displayed. The GSP converts this data to a format meaningful to the 2250.

4.2.1 Initialization Routines

The GSP has an elaborate set of routines to activate the GSP and to establish its environmental characteristics. The highlights of the facilities provided by the initialization routines are:

- (a) Specifications of length and type of input data.
- (b) Allocation of space for the display file that the programmer is to create.
- (c) Scaling to map input data to raster units representing the entire 2250 screen or a small area (a grid) of the screen so that the programmer can create the display in terms of his co-ordinates. If the programmer maps his picture plane to a small area (a grid) of the screen he is provided with two choices: either nothing outside the grid will be displayed (i.e. graphic elements are scissored at the grid limits) or everything outside the grid is still displayed (but graphical elements are scissored at the screen limits).



Dotted lines indicate points/vectors that would not be generated.
 Solid lines indicate points/vectors that would be generated.

Fig 4-3 Scissoring option.

The scissoring option is particularly useful for producing a 'windowing' effect.

- (d) Specification of display mode of input data, whether absolute or incremental.

The programmer can supply all the above control information through three subroutines GSPIN, ICAIN and GCAIN which are quite simple to use.

4.2.2 Graphic Element Generation Routines

Five routines are provided for generating points, line segments, strings of characters and moving the beam position. They are

<u>Subroutine Name</u>	<u>Function</u>
PLINE	Converts input data to 2250 format for displaying line segments between consecutive data points.
PSGMT	Converts input data for plotting line segments between pairs of data points.
PPNT	Converts input data to 2250 format for displaying points at the co-ordinates.
PTEXT	Converts input data to 2250 format for plotting characters
MVPOS	Establishes the starting co-ordinates for the above subroutines.

For example:

```
CALL PSGMT (GCA,XSCoor,YSCoor,XECoor,YECoor,COUNT)
```

A number, given by COUNT, of line segments is

displayed. The starting co-ordinates of the line segments are stored in XSCOOR, YSCOOR and the ending co-ordinates in XECOOR and YECOOR. The subroutine is useful for drawing a number of disconnected line segments. GCA is an array containing the control information (such as scaling, scissoring) for the region in which the line segments are to be drawn.

The programmer has to be cautious in using the subroutine PLINE, whose first data represents the end position of the first line. The initial beam position must be set properly with a call to MVPOS before calling PLINE.

4.2.3 Entity Handling

In the IBM package logical divisions of graphical data are called entities. The complete picture to be displayed must be defined as one image entity which is composed of a number of nested entities. In creating an image entity, a programmer not only defines its nested entities but also its structure through the nesting facilities of the GSP. The nested entities may be named with correlation values which are integers ranging from 1 to 32767. The most important types of nested entities are the controlled entity, the uncontrolled entity and subroutine entity.

A controlled entity is one whose attributes of

visibility and detectability can be specified by the programmer. He can unblank or erase the display of the controlled entity, enable or disable light pen attention on the picture component represented by the entity. A controlled entity may contain uncontrolled entities and subroutine entities but not image entities or controlled entities.

An uncontrolled entity contains display commands for a picture component which always appears on the screen and is not detectable by the light pen. An uncontrolled entity may have controlled entities, other uncontrolled entities, and subroutine entities nested in it but not image entities.

A subroutine entity is similar in concept to the Block in the VISTA package. Its principal use is to display a picture component in more than one area of the 2250 screen in the same image entity (i.e. in the same display). Uncontrolled entities defined in relative mode can be nested in subroutine entities but controlled entities cannot. This restriction implies that a programmer can set a part of a picture component non-detectable by the light pen. Moreover a controlled entity corresponds to a picture component which is fixed on the screen and cannot be displayed at more than one area of the screen. The subroutine entity is parallel to a program subroutine and can be called with the LKSUB subroutine.

The beginning of an entity is specified through a call to the subroutine

```
BELMT (CORRVAL,ELEMENTCODE)
```

where CORRVAL is the correlation value,

and ELEMENTCODE defines the type of entity with

```
ELEMENTCODE = 1  for uncontrolled entity
```

```
              = 2  for controlled entity
```

```
              = 3  for subroutine entity
```

```
              = 4  for image entity
```

In defining an entity, the programmer can define a nested entity within the entity with another call to BELMT before closing the outer entity. The definition of an entity and all its uncompleted nested entities can be closed with a CALL EELMT (CORRVAL), where CORRVAL is the correlation value of the outer-most entity.

Example 1.

```
C  ENTITY 20 IS SUBROUTINE ENTITY WITH UNCONTROLLED ENTITY
C  30 NESTED IN IT
    CALL BELMT(20,3)
    CALL MVPOS(....)
    CALL PPNT(....)
    CALL MVPOS(....)
    CALL PLINE(....)
    .
    .
    .
    CALL BELMT(30,1)
    .
    .
    .
    CALL EELMT(20)
```

In the Example 1, the last statement closes the

definition of both entities.

Subroutine entities can also be nested in subroutine entities. This nesting facility saves the programmer trouble in repeating the definition of basic picture components. Moreover the programmer can assign names to the nested subroutine entities.

Example 2.

```

CALL BELMT(400,3)
ISUBR = 20
ICORVA = 420
SWITCH = 1
CALL MVPOS(      )
CALL LKSUB(ISUBR,ICORVA,SWITCH)
.
.
.
CALL EELMT(400)
C  ISUBR = CORRELATION VALUE FOR THE SUBROUTINE ENTITY
C      LINKED
C  ICORVA = CORRELATION VALUE ASSIGNED TO THE NESTED
C      SUBROUTINE ENTITY
C  SWITCH = 1  UNBLANKS THE NESTED SUBROUTINE ENTITY
C  SWITCH = 2  BLANKS THE NESTED SUBROUTINE ENTITY.
```

In Example 2, the subroutine entity 400 has the subroutine entity 20 nested in it and the nested subroutine entity is named as 420. Thus one picture component can be displayed at different positions but all copies of the picture component have different correlation values for future identification.

A completed image entity can be displayed where CORVAL indicates the image entity to be displayed with CALL EXEC (_____,CORVAL,_____). The programmer may define several pictures as several image entities but only

one image entity can be displayed at a time.

Example 3.

```
CALL  GSPIN(....)
CALL  ICAIN(....)
CALL  GCAIN(....)
CALL  BELMT(1400,3)
.
.
.
CALL  EELMT(1400)
CALL  BELMT(500,4)
.
.
.
CALL  MVPOS(....)
CALL  LKSUB(1400,1401,1)
.
.
.
CALL  BELMT(10,2)
.
.
.
CALL  EELMT(500)
CALL  EXEC(____,500,____)
```

In Example 3, the displayed picture is composed of a subroutine entity 1400 which is called 1401 in the image entity and a controlled entity 10.

4.2.4 Updating Facilities

The GSP package provides three subroutines, DELMT, XELMT, UELMT for modifying the contents of an entity.

The DELMT subroutine identifies an entity that is to be completely deleted from the image entity.

The XELMT identifies an entity whose content is to be extended by subsequent calls to graphic element generation

routines. On completion of extension, the definition of the entity can be closed with the EELMT subroutine. This subroutine is more convenient and simpler to use than the INSERT subroutine in the VISTA package.

The UELMT identifies an entity whose content is to be completely modified by subsequent calls to graphic element generation routines. It is equivalent to DELMT followed with BELMT.

4.2.5 .Attention-handling Subroutines

The subroutine RQATN (Request Attention Information) is provided by the GSP to enable the programmer to obtain attention information at any point in his program. By calling upon this subroutine, the programmer can determine if an attention has occurred and can identify its source.

The general form of the subroutine is

```
RQATN (DEVICE , ARRAY)
```

where DEVICE is an integer with values from 1 to 4 indicating the logical unit used by the package, and ARRAY is a 20-element integer array into which the attention information is to be placed.

The first element of ARRAY indicates if attention has happened and if so, the source of attention.

```
ARRAY(1) = 0    if no attention occurred
           = 2    for light pen attention
           = 4    END key
```


= 8 alphanumeric key

= 16 programmed function keyboard

The other elements of ARRAY contain information such as beam position, the correlation values of the entity when attention occurs, function codes and alphanumeric keyboard codes.

Example 4.

The following sample coding illustrates a procedure that can be followed when it is necessary to wait for an attention before proceeding with program execution.

```

10  CALL EXEC(____,500,____)
    CALL RQATN(1,IARRAY)
    IF (IARRAY(1))80, 20, 30
20  PAUSE
    GO TO 10
30  (process attention)
    .
    .
    .
80  (error - this condition should not occur)

```

ARRAY contains only the correlation value of the entity which is pointed at by the light pen. This entity may be nested in some other entity. The programmer can obtain the type and correlation value of the outer entity

with CALL ROCOR (CORVAL, OUTER, ELEMENTCODE), where CORVAL indicates the correlation value returned by

RQATN

OUTER indicates the correlation value of the entity in which the entity CORVAL is nested

ELEMENTCODE indicates the type of the outer entity.

The chief drawback of this routine is that it provides the correlation value of the entity which is one level higher. A different arrangement of ARRAY in the RQATN subroutine showing all the higher levels might be more convenient to the programmer.

CHAPTER V

HARDWARE/SOFTWARE CONFIGURATION OF A GRAPHICAL DISPLAY SYSTEM

5.0 Introduction

In this chapter, the hardware configuration at the University of Alberta is presented with consideration on the communication link between the graphics sub-system and the main computing system, an IBM 360/67. It describes the operating system in the 360/67 and explains how the main system handles the graphical applications programs. Influences of the 360 hardware and software on the design of the graphical software are considered.

5.1 Display Hardware

A CDC Graphical Remote Interactive Display (GRID) (CDC 1968) has been acquired for display purposes in this project.

The GRID is composed of:

- (a) A cathode ray tube with display rates as follows:

Point Plot

Random positioning	15.2 microseconds
Incremental	5.2 microseconds

Symbol Plot

Random positioning	20 microseconds
------------------------------	-----------------

Tabular mode 6.8 microseconds

When in tabular mode, symbols are drawn in 4.8 microseconds and positioning requires 2 microseconds.

Vector Plot

Vector along the diagonal, $12\sqrt{2}$ inches
34 microseconds

4 inch vector. 15 microseconds

1/2 inch vector. 5 microseconds

Line segments are produced by a line generator.

There is a Refresh command which refreshes the display every 20 msec.

(b) A display controller (a modified CDC 160A computer) Fig 5-1, which besides being a display processor also has logical and arithmetic capabilities. The display controller consists of six logic modules, an analog module, a core memory and a symbol generator.

(i) The interface module serves as a communication network between the S/360 and the display device.

(ii) The display control module contains the control circuitry necessary to obtain display words from the core memory, decode the display words and issue the appropriate command signals to the arithmetic and analog modules.

(iii) The arithmetic module is shared by the processor and display control modules. When the processor is active, the subtracter and logical registers in the arithmetic module are used to perform all processor arithmetic operations. While the equipment is displaying images, the

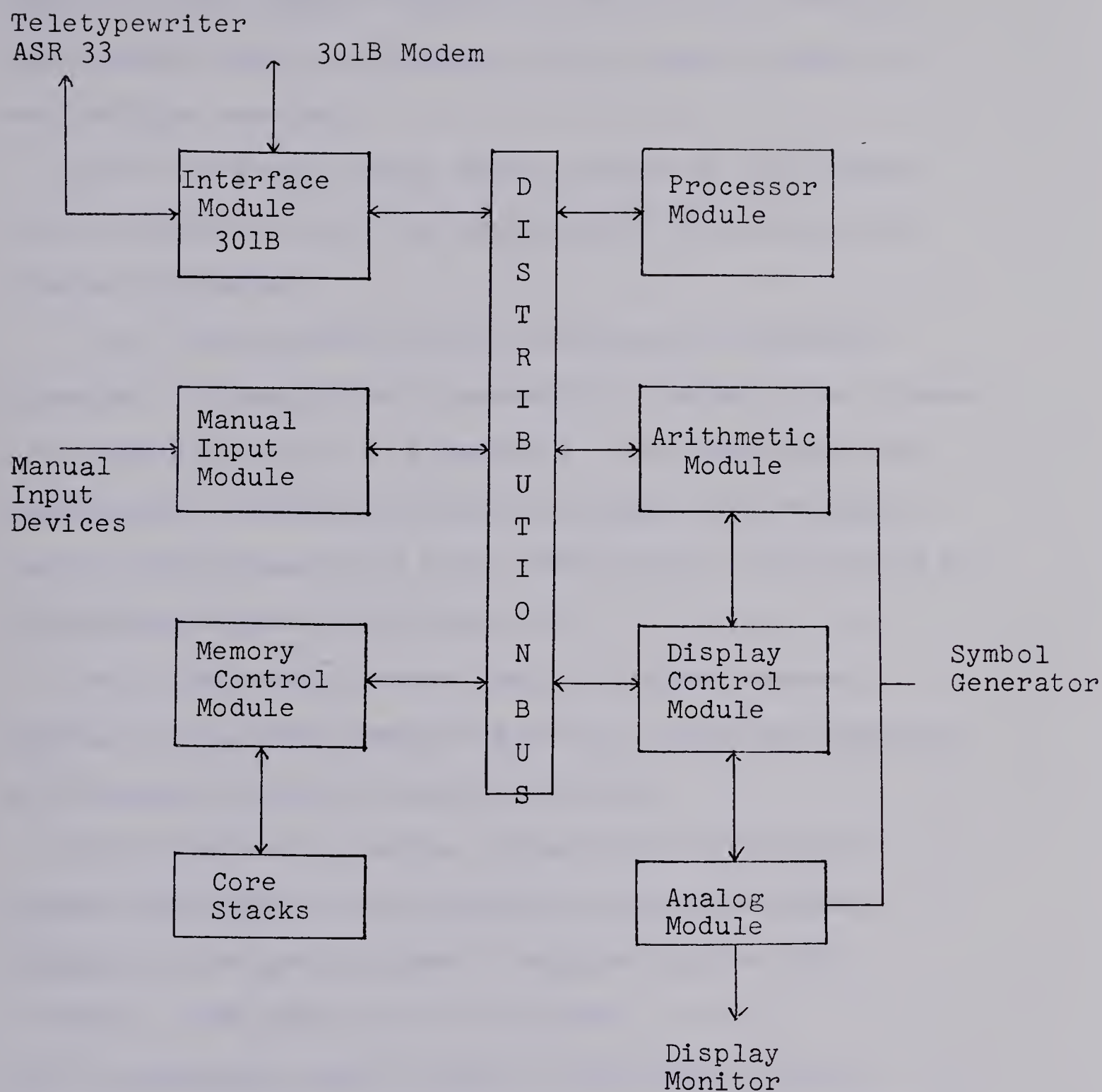


Fig 5-1 Block diagram of the Display Controller.

subtractor and logical registers are used for absolute and relative beam positioning, vector length computation and position scaling.

(iv) The manual input module serves as the communication network between the manual input devices and the display subsystem.

(v) The processor module contains the circuitry necessary to execute the instructions listed in the processor repertoire given in Appendix A. The processor mode instructions include branching, storing, subroutining, logical operations, fixed point addition and subtraction but not multiplication or division.

(vi) The memory control module accepts control signals to initiate memory read/write cycles and supplies all necessary address control circuitry.

(vii) The analog module contains the circuitry to convert the digital base position, vector and symbol signals to the analog signals required by the CRT.

(viii) A 4K store of 12 bit words.

(c) An operator control panel is used for off-line program entry, program checkout and general maintenance.

(d) An alphanumeric and function keyboard. The alphanumeric portion of the keyboard includes the alphabet set, digits and 25 special characters. The keyboard can be enabled or disabled by the display program. When the keyboard is enabled, each key, when pressed, provides a

unique code to be interpreted by the processor module.

The function keyboard includes special edit keys, 10 function keys, 4 status switches (allowing 16 combinations) and an INT (Interrupt) key.

(e) A light pen. Light pen enable or disable command words may be placed at any number of places in the display program so that some items on the screen can be set detectable while some items are not sensitive.

5.2 Data Transmission Between the Graphical Subsystem and the Main Computer

The modified CDC 160A computer in the GRID system has limited functional capability. For example it has no provision for multiplication (except multiplication by decimal 10 only), division, right shift and indexing. Its major responsibility lies in executing display instructions and in performing some household tasks eg. message assembling and tracking. Moreover, display programs normally require a large data base. With a core storage of 4K, it is impossible for the GRID to act as a stand-alone device for non-trivial applications. This calls for a larger computer for involved computations and data bases. An IBM 360/67 is used for this purpose. A data link is necessary to connect the GRID computer with the 360/67.

A half-duplex telecommunications line of 40.8 kilobits/sec. having W.E. type 301B modems is used. An

IBM 2701 Data Adaptor Unit is used to support the telecommunication. The 2701 Data Adaptor Unit is linked to the 360/67 via a 2870 Multiplexor Channel. The 360/GRID configuration is shown in Fig 5-2.

The frequency of communication and the amount of data in each transmission vary from application to application. But since the GRID operator is expecting a reply at the console, a response close to human reaction time is necessary. The amount of data transmitted may be as high as 4K as in an entire core dump. A medium-high transmission speed is thus required. Lower interface cost and higher transmission speed could have been obtained if the GRID unit were linked directly to a channel of the 360/67 instead of adopting a telecommunication line. The telecommunication link was chosen because the problems of equipment compatibility were then simpler. The same interface would theoretically be compatible with any future replacement of the System/360.

Many existing data communication links use control procedures and formats that have been optimised for particular devices, systems and applications. They are thus unsuitable for other devices and applications. For example the IBM Synchronous Transmitter Receiver (STR) family of transmission devices uses a transmission code limited to 64 data characters, so that it cannot easily accommodate eight bit data byte used in the System/360. Most teletype-

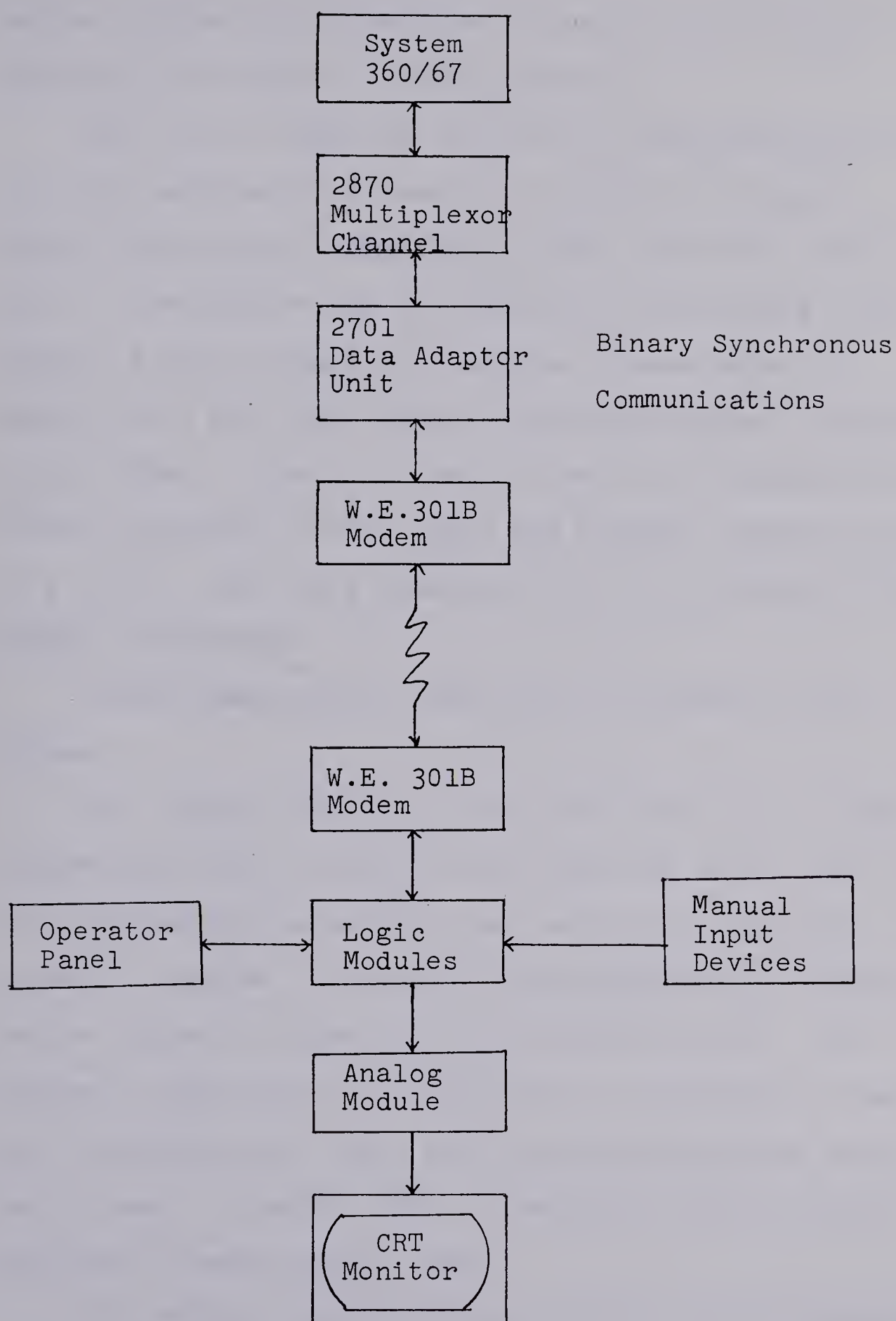


Fig 5-2 360/GRID Configuration

writer systems do not provide for error detection and correction by automatic retransmission.

The 2701 is supplied in various configurations, but for this application it must be equipped to support Binary Synchronous Communication, BSC (Eisenbies 1967). BSC is a definitive set of conventions describing every aspect of the automatic synchronous transmission of digital data over half-duplex (non-simultaneous) communication links. Provision has been made for communication between different device types and between computer processing units. BSC gives compatibility for a possible future change of hardware.

A BSC communication link can be in either of two states:

(a) Control state, in which the direction of data transmission and the role of the stations on the line during the forthcoming transmission are established (whether they transmit, receive, or monitor the transmission). There are two types of operation in the control state: the contention mode which is restricted to operation on two-point communication lines and the supervised mode which can be used to operate both on two-point and dedicated multipoint communication lines.

(b) Message transfer state, in which the principal function is the transmission of data.

Since the communication between S/360 and GRID is

only point-to-point, the contention mode of operation is used. Both the GRID and S/360 are always in a 'READ' state ready to receive transmissions unless servicing a 'WRITE' command. Either the 360 or GRID can initiate a transmission by 'bidding', it signals its intention and requests permission to transmit by sending a synchronizing pattern \emptyset and the enquiry (ENQ) character. The other station is then obliged to reply, indicating whether or not it is able to receive a transmission by using the affirmative (ACK) or negative (NAK) reply signals.

To provide for the situation in which both stations simultaneously bid to become the transmitter, one of the stations is denoted the primary station and the other the secondary station. In this application the S/360 is the primary station and GRID the secondary station. If both stations attempt to transmit simultaneously, the primary station will transmit first.

When an affirmative reply to a request is obtained, the enquiring station gets into the message transfer state and sends the following message

\emptyset STX ESC text ETB bcc,

where STX indicates the data between ESC (the escape character) and ETB (end of text block character) is the text to be transmitted. A block check character (bcc) is generated from the data in an identical fashion at both the transmitter and the receiver, and the receiver compares the received bcc

with the one that is generated from the received data. If not identical, an error has occurred; if identical the transmitted text block is assumed to be error free. After the message is transmitted, the transmitter awaits a reply for successful transmission. If the reply is negative (NAK), the transmission is repeated a limited number of times before calling the error routine which reports hardware trouble. If the transmission is successful, the transmitting station sends the next block if there are more than one text block in the complete message, or the Ø EOT command (end of transmission) when all the text blocks are transmitted successfully. The communication line enters the control state when the EOT is signaled. For example, assume the 360 attempts to transmit two segments of DISPLAY FILE to be placed at two areas of the GRID core. The transmission can be shown in Fig 5-3.

Legend: Ø Synchronizing pattern,
* Change of direction in transmission,
** No reply expected.

5.3 Operating System on S/360

A multiprogramming with a variable number of tasks (MVT) configuration of S/360 Operating System (OS) (IBM Form Y28-6659-2, IBM Form C28-6535-1) is run on the S/360. Multiprogramming of system functions and up to 15 jobs is allowed.

<u>Type</u>	<u>Format</u>	<u>Remarks</u>
Enquiry	ØENQ*	360 requests for transmission to GRID.
Reply	ØACK0*	Even affirmative acknowledge reply to second and all even-numbered text blocks.
Text	ØSTX ESC A1 A2 ETB bcc*	First text block to GRID. A1 indicates beginning address of first segment in GRID core. A2: (end address +1) of first segment in GRID core.
Reply	ØACK1*	Odd affirmative acknowledge reply from GRID to first and all odd numbered text blocks.
Text	ØSTX ESC text ETB bcc*	Second text block. The first segment transmitted
Reply	ØACK0*	
Text	ØSTX ESC A1 A2 ETB bcc*	Third text block. A1: beginning address of 2nd segment in GRID core. A2: End address +1 of the second segment in GRID core
Reply	ACK1*	
Text	ØSTX ESC text ETB bcc*	Fourth text block. 2nd segment transmitted.
Reply	ØACK0*	
End	ØEOT**	

Fig 5-3. The transmission of two segments of DISPLAY FILE from S/360 to GRID.

The programs that compose the Operating System are classified as a control program and processing programs. The control program has three major functions: job management, task management and data management.

The job management functions are:

1. scanning the input stream to identify the control statement, interpreting and analyzing the control statements, preparing the necessary control tables that describe each job to the system;
2. allocation of I/O devices: ensuring that all necessary I/O devices are allocated;
3. selecting jobs for execution on a priority basis;
4. transmission of input data onto, and user output from a direct-access device;
5. communication between the operator and the system.

Task management is primarily a supervisory function which controls all the tasks in the system and is usually called the supervisor. This portion of the Operating System includes allocating some system resources to tasks, passing CPU control to the proper routine when a control program service is requested for a task, determining the priority among the tasks, and passing control to routines of the tasks.

Data management routines perform operations associated with input and output devices. This includes allocation of space on direct-access volumes, storing, naming and cata-

logging of files and scheduling of I/O operations.

The processing programs consist of language translators (such as high level language compilers and assemblers), service programs (such as the linkage editor, Sort/Merge programs, utility programs) and applications programs which are written by the user.

The programmer may write an applications program in FORTRAN using a FORTRAN graphical subroutine package and submit the job to S/360. Up to 15 jobs can be competing concurrently under MVT. Each job, consisting of one or more processing programs (steps), is selected for execution from an input work queue by priority. The task's current state determines the task's readiness to use the CPU. If the task can make immediate use of the CPU, it is ready. If the task is using the CPU, it is active. Otherwise the task is in wait state. A task can enter the wait state directly through macro instructions or indirectly when it waits for an input record. The program being executed can be interrupted, possibly because it contains a request for supervisor service or possibly because an input/output operation has been completed for an entirely different task. When an interrupt occurs, control is passed to the interruption-handling portion of the supervisor which analyses the interruption, based on the control information passed to it at the time of the interruption and takes the appropriate action. If an active task, for example task A, is

interrupted, it will only get control back (after the supervisor processes the interruption) if all other ready tasks are of equal or lower priority. If however, a higher priority task, task B, is now ready, it is given control, regardless of the point at which task A is interrupted. Task A is still in ready state.

Since the graphical application requires fast response, it is assigned high priority. When the graphical task is given control, through calls to the graphical subroutine package the graphics applications program prepares GRID display commands in a display file. When the display file is completed it is written into the GRID core by calling transmission routines and the program relinquishes control to the 360 job management routines which will schedule the next task with the highest priority. The display applications program in 360 remains in wait state until a message is received from GRID. When such a message arrives from GRID, the execution of the program presently under control in S/360 will be stopped. The supervisor will service the interrupt and set the display applications program into ready state and pass control to it if it has the highest priority among the ready tasks. The display program will then delete or modify the display file or generate new graphical commands in the display file. The new display file will again be transmitted to GRID for display and the display applications program in S/360 returns to the wait

state. One of the other tasks in ready state will gain control.

5.4 Influences on the Design of the Graphics Software

The graphical display is most useful in that the on-line user can work in a conversational mode as in computer aided design, or in an iterative procedure as in on-line engineering design. In both cases, the GRID console operator uses the manual input devices frequently. He can:

- (1) Pick displayed items with the light pen.
- (2) Press function keys which have some meaning assigned within the applications program.
- (3) Type strings of alphanumeric characters.
- (4) Have vectors drawn between points on the screen.
- (5) Indicate positions on the screen.

The GRID subsystem has to rely on S/360 for processing manual input attentions and other computations. Since S/360 has to give high priority to the graphical tasks, the processing of S/360 may be severely affected if every manual input attention causes an interrupt to 360.

In order to reduce the associated overhead in interrupt-servicing to provide more productive time for other System/360 uses, the number of interrupts on the System/360 should be reduced to a minimum. Each attention requested by the manual input devices of the GRID subsystem will therefore not be routed to the System/360 as an

interrupt. In general several attentions are required to supply the necessary parameters for a desired function. For example, to merely draw a line between 2 positions A and B on the screen, the GRID operator may have to

1. point at the command word CONNECT on the screen with a light pen,

2. point the light pen at position A so that the (X,Y) co-ordinates of A can be picked when the scanning pattern is displayed,

3. press the SCAN key on the programmed function keyboard,

4. press the HERE key on the programmed function keyboard,

5. point the light pen at B,

6. press the SCAN key,

7. press the HERE key,

8. press the SEND key on the console.

A small display supervisor which is resident in the GRID core is thus necessary to collect all the input parameters caused by each of the above attentions, assemble a message with the parameters and return control to the display processor after servicing each attention. The GRID supervisor must possess a set of routines to interpret some of the keyboard attentions such as the SCAN, HERE and SEND keys. When the SEND key is pressed, the GRID supervisor sends the assembled message to S/360 and

only then produces an interrupt on S/360. The message assembling facilities of the GRID supervisor help to avoid interruption on the S/360 to perform trivial operations.

Another problem confronted is the transmission of the display file. The modified display file (in the above example, with the addition of a line joining A and B) has to be transmitted to GRID for display. Very often only small sections of the display file have been changed. It will lengthen the transmission time if the whole display file is transmitted. With the transmission speed of 40.8 kilobits/sec., it takes more than 1 sec. to transmit the 4K 12 bit words to GRID. However, as can be seen in section 5.2, the transmission of one segment of the display file requires two text block transmissions. The first transmission sends the beginning and ending addresses of the segment of the display file and the second sends the display file between the two addresses. Moreover S/360 has to wait for an affirmative reply from GRID before and after each text block transmission. Thus in the transmission of a large number, say more than 10, of small segments of the display file the overhead in executing the 360/GRID input/output routines may increase considerably. Reducing the number of segments to be transmitted by enlarging the segments to include others will decrease the overhead but increase the transmission time. A transmission subroutine

directed towards the applications programmer is indispensable to initiate the transmission to GRID, wait for a message from GRID and interpret the message. This subroutine should also decide the optimal number and length of segments of display file to be transmitted.

CHAPTER VI

GRIDSUB, A GRAPHICAL SUBROUTINE PACKAGE

6.0 Introduction

A FORTRAN graphical subroutine package, GRIDSUB (GRID SUBroutine Package), has been defined and developed by the author for the 360/67 - GRID system. The first version of the package is ready for release to the user public. In this chapter the specification of GRIDSUB is presented. Some of the special features of the package are described in comparison to the VISTA and IBM packages. In conclusion the package is evaluated with reference to some actual applications.

6.1 Specification of GRIDSUB

The package GRIDSUB contains subroutines for displaying graphic forms on the GRID screen and for controlling communication between the applications program and the GRID user. The applications programmer is concerned with creating, displaying and modifying a drawing on the screen. He defines graphic elements, which are points, lines and characters, in the form of input data to subroutines which convert the data into GRID display commands.

In creating a picture on the screen, the applications programmer not only defines its elements but also its structure. He has to divide the graphical data into discrete units, each of which corresponds to a picture

component. Each discrete unit of graphical data is called a block and can be identified by a number in the range of 1 to 511.

The subroutines can be classified according to their functions:

- (a) Block handling
- (b) Generation of graphic elements
- (c) Display of blocks
- (d) Modification of a displayed picture
- (e) Transmission and decoding facilities

The definition of a block is opened by a call to the BLOCK subroutine which attaches an integer, the block number, to the graphical data which is to be contained in the block. The content of a block is defined by calls to graphic element generation subroutines:

VECTOR, which displays a vector

DLINE, which displays a series of connected line segments,

MOVE, which moves the beam without unblanking it,

POINT, which displays a point,

TEXT, which displays a string of characters,

or ADDBLK, which provides nesting of blocks by inserting into the block a previously defined block.

Finally the block definition is closed by a call to ENDBLK.

Each block will have an origin, the point at which

the definition begins. The programmer can specify co-ordinates of the graphic elements in two modes:

absolute (relative to block origin)

relative (relative to the beam position after the previous element of the definition)

The screen is assumed to be a rectangular grid of 1024×1024 raster points. A raster unit which is approximately 0.0117 inches, is the distance between a pair of neighbouring raster points.

Suppose the NOT gate in Fig 6-1 is to be defined as a block.

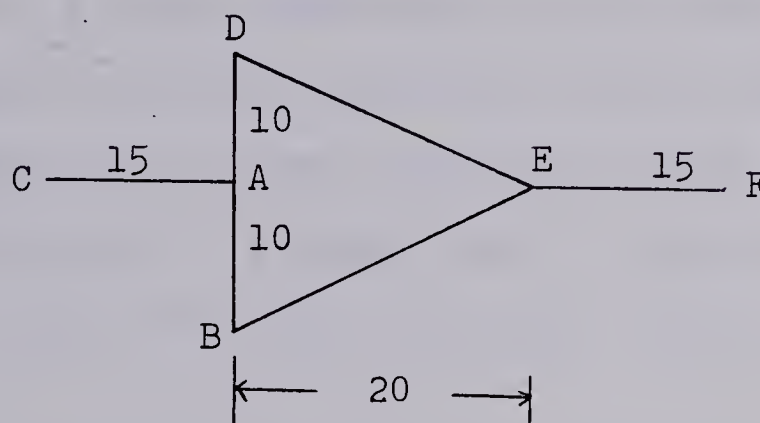


Fig 6-1 A NOT gate.

Assume that the absolute mode is used. If the definition begins at point A, A is taken automatically as the block origin (0,0). The co-ordinates of the rest of the points will be B(0,-10), C(-15,0), D(0,10), E(20,0) and F(35,0). Actual screen co-ordinates are not used in block definitions.

A completed block may be displayed with the DISPLY subroutine at a number of positions on the screen. Abso-

lute screen co-ordinates are used to indicate the origin of the block to be displayed. An identification number ID is used to distinguish copies of the same block displayed at different positions. A block can be displayed at not more than 64 positions with values of ID ranging from 0 to 63. An attempt to display a block whose definition has not been closed will be ignored.

A set of routines is provided to modify a displayed picture. The position of a block displayed on the screen can be changed by a MOVEBK call. Blocks and nested blocks can be erased with the ERSBK and ERSNBK subroutines. A specific copy of a block displayed on the screen can be blanked out with the ERSID subroutine but all other identical copies of the same block will not be affected. Any blanked block can be brought back to active state and becomes visible with the reset routines, RSTBK, RSTNBK and RSTID.

It is advisable to destroy blocks which are not wanted to save space both in the S/360 and GRID core. Blocks can be destroyed with the DELBLK subroutine. A specific copy of a block can be destroyed with DELID subroutine. A call to FREEBK will destroy a block in GRID core while the graphical data relating to the block is still preserved in S/360. The applications programmer can set the light pen detectability of a block with the subroutine LP.

The GRIDSUB prepares the GRID display commands in a display file in S/360. This file of GRID commands must be transmitted to the display device to be executed before the image can appear on the screen. The applications program can initiate the transmission of the display file by calling TRANMT. The TRANMT subroutine transmits to GRID the whole display file or portions of the display file which have been modified and puts the applications program into a wait state. The display of the image still continues in GRID. The GRID console user can now study the displayed image and indicate his desired actions through manual input devices. The set of all allowable sequences of attentions for any application constitutes the "command language" of that program. The attentions caused by the manual input devices are processed and assembled into a message by a GRID supervisor which is resident in the GRID core. When the message is completed, the GRID console user has to press a SEND key. On sensing the attention from the SEND key, the GRID supervisor initiates the GRID input/output routine to transmit the message to S/360. The TRANMT subroutine gets back into the active state and converts the message into programmer-oriented parameters such as type of attention, block numbers. The parameters are stored in two arrays MSGTBL, MESSAG in a common area and the FORTRAN statement following CALL TRANMT in the applications program gains control.

A group of decoding routines are also provided to interpret the arrays MSGTBL, MESSAG. The routines DLPEN, DFKEY, DALPHA check if a particular operation of the GRID console user is a light pen, function key or alphanumeric keyboard attention respectively. The decoding routines DVECT and DPOINT convert line segments or points detected or drawn on the screen into arrays of X and Y values. The DEND subroutine tests for the "end of message" or the pressing of the SEND key.

6.1.1 Arguments Used by Many of the Subroutines

In order to avoid repeated description of arguments that are common to several subroutines in the package, a general description of such arguments is presented here.

(a) X and Y

Type of variable: INTEGER*4

X and Y indicate co-ordinates in raster units.

INTEGER variable, INTEGER constant with length 4, or INTEGER arithmetic expression can be used as actual arguments. X and Y may indicate the position of the CRT beam, beginning position of the first character in a string of characters or end position of a line segment. The pair (X,Y) is considered as X and Y increments in relative mode and can have any value in the range of -1023 to 1023. When in absolute mode, the pair (X,Y) is interpreted as a position relative to the block origin and can also have

values in the range of -1023 to 1023 subject to the condition that X or Y is not more than 1023 raster units away from the last beam position. For example, assume the initial beam position is P(600,400) with respect to the block origin. A CALL VECTOR which draws a vector to Q(-500,20) with respect to the block origin is not acceptable because the vector will be outside screen limits no matter how the block is displayed.

(b) RELBM

Type of variable: LOGICAL*1

RELBM is always used in conjunction with X and Y, and indicates the mode of block definition.

If RELBM = .FALSE., the co-ordinates (X,Y) are taken to be relative to the block origin.

If RELBM = .TRUE., the values (X,Y) are taken to be increments relative to the last beam position.

(c) BLINK

Type of variable: LOGICAL*1

If BLINK = .TRUE., the graphic element (point, vector, or character) defined by the graphic element generation routine of which BLINK is an argument will, on displaying, be shown blinking.

If BLINK = .FALSE., the graphic element will be shown without blinking.

(d) DASH

Type of variable: LOGICAL*1

If DASH = .FALSE., a solid line will be displayed.

If DASH = .TRUE., a dashed line will be displayed.

(e) NUMBER

Type of variable: INTEGER*4

NUMBER is used by the programmer to name a defined block. NUMBER can be a positive INTEGER constant, INTEGER variable or INTEGER arithmetic expression in the range 1 to 511. The value must be unique for each block defined. The block number of a block that has been destroyed by the subroutine DELBLK can be reused to name a new block.

(f) ID

Type of variable: INTEGER*4

ID serves as an identifier to distinguish copies of the same block displayed at different positions. ID may be an INTEGER constant, INTEGER variable, or INTEGER arithmetic expression in the range of 0 to 63. The value must be unique for each copy of the block.

6.1.2 Block Definition

(a) BLOCK (NUMBER)

NUMBER is defined in section 6.1.1

The definition of a block is opened and henceforth the block will be referred to by the value of NUMBER

Possible errors*:

* When an error is encountered, the subroutine prints the error message and ignores the call.

- (i) Previous block not ended
- (ii) NUMBER zero or negative
- (iii) Number of blocks exceeds 100
- (iv) Doubly defined block numbers

(b) ENDBLK

This subroutine which has no arguments closes the definition of a block. Definition of a block must be closed with a call to ENDBLK before the next block can be defined.

Possible errors:

- (i) BLOCK FILE full.

6.1.3 Graphic Element Generation

(a) MOVE (RELBM , X , Y)

RELBM, X and Y are defined in section 6.1.1

The beam is moved from the current beam position, (X_i, Y_i) , to $(X_i + X, Y_i + Y)$ if RELBM is .TRUE. or the beam is positioned without unblanking at (X, Y) with respect to the block origin if RELBM is .FALSE.

Possible errors:

- (i) A block has not been opened.
- (ii) The X or Y increment is outside the range -1023 and 1023.

(b) VECTOR (RELBM, X,Y, BLINK, DASH)

All arguments are defined in section 6.1.1

This subroutine creates a line from the current

beam position (X_i, Y_i) to $(X_i + X, Y_i + Y)$ if RELBM is .TRUE. or to (X, Y) with respect to the block origin if RELBM is .FALSE..

Possible errors:

(d) A block has not been opened

(ii) The X or Y increment is outside the range -1023 and 1023.

(c) DLINE (RELBM, RELBM1, AX, AY, N, BLINK, DASH)

Type of variables or arrays: LOGICAL*1 RELBM1
 INTEGER*2 AX (array),
 AY (array).
 INTEGER*4 N

AX and AY are arrays of at least N elements. A line is drawn through each of the N points which are represented by the data stored in AX and AY. The first data point $(AX(1), AY(1))$ is the beginning point of the first line segment. The beam is moved to the position indicated by $(AX(1), AY(1))$ before the line segments are drawn.

If RELBM = .FALSE., $(AX(1), AY(1))$ is the position, relative to the block origin, of the beginning point of the first line segment.

If RELBM = .TRUE., $(AX(1), AY(1))$ represents the increments relative to the current beam position, of the position of the beginning point of the first line segment.

If RELBM1 = .FALSE., $(AX(I), AY(I))$, $2 \leq I \leq N$, is the end point of the $(I-1)$ th line segment, relative to the block

origin.

If RELBM1 = .TRUE., (AX(I), AY(I)), $2 \leq I \leq N$, represent increments of the Ith point relative to the (I-1)th point.

Possible errors:

- (i) A block has not been opened.
- (ii) The X or Y increment is greater than 1023

(d) POINT (RELBM, X, Y, BLINK).

All arguments are defined in section 6.1.1.

A point is drawn at (X,Y) with respect to the block origin if RELBM is .FALSE.

A point is drawn at ($X_i + X$, $Y_i + Y$) if RELBM is .TRUE. and the initial beam position is at (X_i, Y_i).

Possible errors:

- (i) A block has not been opened
- (ii) The X or Y increment is outside the range -1023 and 1023.

(e) TEXT (RELBM, X,Y,N, CHAR, LARGE, BLINK)

The arguments RELBM, X, Y, BLINK are defined in section 6.1.1.

Type of variable: INTEGER*4 N

INTEGER*2 CHAR(43)

LOGICAL*1 LARGE

Values of X and Y give the position of the centre of the first character.

If N is positive, N characters are displayed in

the +X direction.

If N is negative $|N|$ characters are displayed in the +Y direction i.e. from bottom to top and rotated through 90° . The maximum value of $|N|$ is 86.

The programmer can use a character string enclosed in apostrophes, a character string in H FORMAT, or an INTEGER*2 variable or array as the actual argument for CHAR. If an INTEGER array is used, each element (half-word) must contain 2 characters and the maximum dimension of the array is 43. The GRID character set is slightly different from that provided on the 029 card punch machine. The conversion table is shown in Fig 6-2.

If LARGE = .FALSE., characters are approximately
10.7 (wide) x 14.2 (high)
raster units. The spacing
between neighbouring characters is 12 raster units.

If LARGE = .TRUE., characters are approximately
13.6 (wide) x 17.9 (high)
raster units. The spacing
between neighbouring characters is 16 raster units.

Possible errors:

- (i) Block has not been opened.
- (ii) The X or Y increment is outside the range -1023 and 1023.

FORTRAN Symbol	GRID Symbol	FORTRAN Symbol	GRID Symbol	FORTRAN Symbol	GRID Symbol
:	:	X	X	!	↓
1	1	y	y	>	>
2	2	Z	Z	+	+
3	3	,	,	A	A
4	4	({	B	B
5	5	@	→	C	C
6	6	—	—	D	D
7	7	&	^	E	E
8	8	-	-	F	F
9	9	J	J	G	G
0	0	K	K	H	H
=	=	L	L	I	I
#	≠	M	M	<	<
"	≤	N	N	.	.
%	%	O	O)	}
Space	Space	P	P	¢	≥
/	/	Q	Q	┐	┐
S	S	R	R	;	;
T	T		v		
U	U	\$	\$		
V	V	*	*		
W	W	?	↑		

Fig 6-2. Conversion Table for FORTRAN and GRID Character Sets.

(iii) N equals zero

(iv) Too many characters.

(f) ADDBLK (NUMBER, RELBM, X,Y)

All arguments are defined in section 6.1.1.

A previously defined block, designated by NUMBER, is used in the definition of the block under construction. The origin of the included block is taken as (X,Y).

Note that the beam position after displaying the nested block is reset automatically to the initial position before the ADDBLK was called.

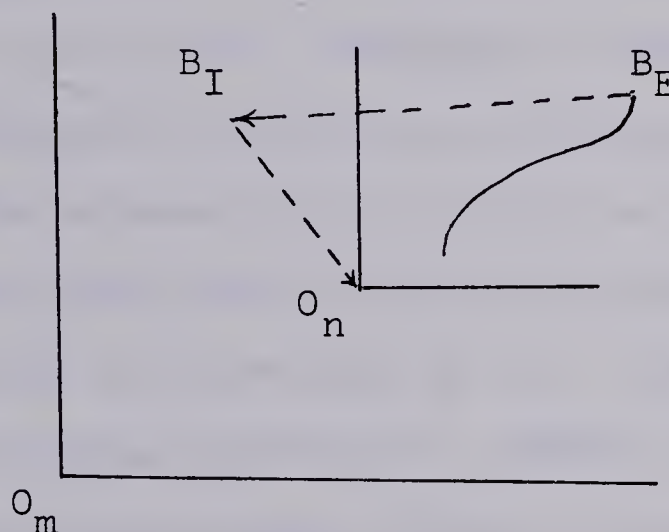


Fig 6-3.

O_m = Origin of the main block under construction

O_n = Origin of the nested block

B_I = Initial beam position before calling ADDBLK

B_E = Beam position at the end of display of the nested block.

The beam is transferred from B_E to B_I after displaying the nested block.

The maximum number of levels of nesting is seven. Unpredictable display will result when the number of levels of nesting exceeds seven.

Possible errors:

- (i) Nested block NUMBER is not defined
- (ii) A block has not been opened
- (iii) The Block File Table is full

6.1.4 Block Display

DISPLY (NUMBER, ID, ABSX, ABSY)

NUMBER and ID are defined in section 6.1.1.

Type of variable: INTEGER*4 ABSX, ABSY

A previously defined block designated by NUMBER is displayed on the screen with its origin at the absolute screen position (ABSX, ABSY) $0 \leq \text{ABSX} \leq 1023$, $0 \leq \text{ABSY} \leq 1023$. The copy of the block is identified by ID. NUMBER and ID are useful for processing attentions. NUMBER indicates the type of block and ID distinguishes between copies of the same block. If the NUMBER ID combination has been used in a previous call to DISPLY, the present call will be ignored.

Possible errors:

- (i) NUMBER outside range
- (ii) ID outside range
- (iii) Block identified by NUMBER not defined
- (iv) Block not closed
- (v) Screen co-ordinates outside range

- (vi) Display File Table is full
- (vii) Display File is full

6.1.5 Modification of Display

(a) MOVEBK (NUMBER, ID, ABSX, ABSY)

NUMBER and ID are defined in section 6.1.1

Type of variable: INTEGER*4 ABSX, ABSY

This subroutine moves a displayed block designated by NUMBER and ID to a new screen position (ABSX, ABSY). The old block will disappear and the new displayed block will have the same NUMBER and ID as the old block. (ABSX, ABSY) must be in absolute screen co-ordinates in the range of 0 to 1023.

Possible errors:

- (i) The block identified by NUMBER and ID is not displayed
- (ii) Screen co-ordinates outside range

(b) ERSBK (NUMBER)

NUMBER is defined in section 6.1.1

The block designated by NUMBER is erased from the screen. If a block is displayed at more than one place, all copies of the block are erased. However the definition of the block is still kept in the GRID core. The displaying of the block can be made active again with the RSTBK subroutine.

Possible errors:

(i) The block identified by NUMBER is not defined.

(c) RSTBK (NUMBER)

NUMBER is defined in section 6.1.1

The erased block, designated by NUMBER, is unblanked.

Possible error:

(i) The block identified by NUMBER is not defined.

(d) ERSID (NUMBER, ID)

NUMBER and ID are defined in section 6.1.1

The copy of a block designated by NUMBER and ID is erased. All other displayed blocks with the same NUMBER but different ID's will not be affected.

Possible error:

(i) The block identified by NUMBER and ID is not defined.

(e) RSTID (NUMBER, ID)

NUMBER and ID are defined in section 6.1.1

An erased copy of a block is unblanked.

Possible error:

(i) The copy of the block identified by NUMBER and ID is not defined.

(f) ERSNBK (NUMBER, NESTBK)

NUMBER is defined in section 6.1.1

Type of variable: INTEGER*4 NESTBK

The nested block with block number NESTBK in a main block called NUMBER is erased from the main block. All the copies of the block displayed on the screen will be affected.

Possible errors:

(i) The block identified by NUMBER does not contain a nested block called NESTBK

(ii) The block identified by NUMBER is not defined.

(g) RSTNBK (NUMBER, NESTBK)

The arguments are defined exactly as in ERSNBK.

The erased nested block NESTBK of a main block, NUMBER, is unblanked again. All copies of the block NUMBER displayed on the screen will be affected.

Possible errors:

(i) The block identified by NUMBER does not contain a nested block called NESTBK

(ii) The block identified by NUMBER is not defined.

(h) DELBLK (NUMBER)

NUMBER is defined in section 6.1.1

The designated block will be destroyed from the GRID core memory and S/360 workspace. All information concerning the block is lost. The block name, NUMBER, is also cancelled from the block name list. The programmer

may use NUMBER to designate a new block.

Possible error:

(i) The block designated by NUMBER is not defined.

(i) DELID (NUMBER, ID)

NUMBER and ID are defined in section 6.1.1

The copy of the block designated by block number NUMBER and identification ID is destroyed permanently from the display. This subroutine is different from ERSID in that this specific copy of the block cannot be unblanked again.

Possible error:

(i) The copy of the block, identified by NUMBER and ID, is not defined.

(j) FREEBK (NUMBER)

NUMBER is defined in section 6.1.1

The designated block is removed from the GRID core but its definition still remains in the S/360 workspace.

(k) LP (NUMBER, ID, DETECT)

NUMBER and ID are defined in section 6.1.1

Type of variable: LOGICAL*1 DETECT

This subroutine allows the FORTRAN programmer to change the light pen detectability of a specific displayed copy of a block.

If DETECT = .FALSE., the copy ID of the block NUMBER is
light pen disabled.

If DETECT = .TRUE., light pen attention on the copy ID of the block NUMBER is recognised.

6.1.6 Transmission and Decoding routines:

(a) TRANMT*

This subroutine has no arguments.

The TRANMT (transmit) has two main functions.

First it determines which parts of the display file have to be transmitted to GRID. It builds a maximum of 6 variable-length buffers and passes pointers of these buffers to the internal input/output routine.

Secondly, when control is returned to TRANMT from the internal input/output routine (a message has been received from GRID) TRANMT checks to ensure the transmission to GRID and from GRID was successful. If not, TRANMT terminates the applications program. If transmission was successful, TRANMT builds a table MSGTBL and decodes the message into an array MESSAG and returns control to the applications program. MSGTBL and MESSAG are arrays of type INTEGER*2, in a labeled common area. The specification of MSGTBL and MESSAG is given in Appendix B.

The FORTRAN programmer can either decode the operator's message directly from MSGTBL and MESSAG or use the message decoding routines.

* Coded by F. Jacobsen.

(b) Decoding routines*:

(i) Arguments:

All the arguments of the decoding routines except STRING, X,Y,* are INTEGER*4 variables. STRING, X, Y are INTEGER*2 arrays. Values supplied by the applications program for INTNO and MAX may be constants instead of variables.

INTNO specifies which operator action is to be queried. Hence $1 \leq \text{INTNO} \leq 20$.

The last argument of a calling sequence to each of the decoding routines is always of the form &n. If the operation action agrees with the type of attention queried, the arguments are filled in and control is transferred to the statement labeled n. If not, control is passed to the next statement following the calling sequence.

(ii) DLPEN (INTNO, IX, IY, TYPE, ID, BLK, *)

If the operator action is light pen attention, the X, Y co-ordinates of the light pen hit are filled in IX, IY. TYPE specifies what type of element was picked by the light pen: 0 for a point, 1 for a symbol and 2 for a vector. ID specifies the ID ($0 \leq \text{ID} \leq 63$) assigned to the block by the DISPLY call. BLK is an array of 8 elements. BLK(1) contains the block number of the outer block. BLK(2) contains the first-level, nested block number. BLK(3)

* Specified and coded by F. Jacobsen

contains the second-level nested block number and so on. Unused elements in BLK are set to zero.

(iii) DFKEY (INTNO, STATUS, KEY, *)

The value of the status key (1 to 15) when a function key or interrupt key was pressed is stored in STATUS. KEY has a value 0 to 9 if a function key was pressed, or -1 if the interrupt key was pressed.

(iv) DALPHA (INTNO, IX, IY, NUM, MAX, STRING, *)

If the attention is from the alphanumeric keyboard, IX, IY contain the screen co-ordinates of the first character pressed. MAX specifies the maximum number of halfwords which are to be filled with characters. Maximum number of characters that can be returned is 2*MAX. NUM specifies the actual number of halfwords returned. The number of characters returned may be 2*NUM or 2*NUM-1. STRING is the array which is filled with the characters returned.

(v) DVECT (INTNO, NUM, MAX, X, Y, *)

(vi) DPOINT (INTNO, NUM, MAX, X, Y, *)

MAX specifies the maximum number of (X,Y) co-ordinates that are to be passed back to the applications program. NUM specifies the actual number of (X,Y) co-ordinates which are passed back to the applications program in a DPOINT or DVECT call. In a DPOINT call, NUM pairs

of co-ordinates which are contained in X,Y are specified. In a DVECT, NUM-1 connected vectors are specified, (X(1), Y(1)) specifies the beginning of the first vector and (X(NUM), Y(NUM)) specifies the end of the last vector. (X(I), Y(I)) $2 \leq I \leq \text{NUM}$, specifies the end of the (I-1)th vector and the beginning of the Ith vector.

(vii) DEND (INTNO, *)

Tests if an operator action is the pressing of the SEND key.

Appendix D shows an example using the decoding routines.

6.2 Discussion on some of the features of the package compared with the VISTA and IBM subroutines.

6.2.1 DLINE

In the GRIDSUB package, graphical data of discrete components of a picture is called a block which is equivalent in concept to the BLOCK in the VISTA package and the subroutine entity of the IBM package. Since a block is to be displayed at several positions, the display commands generated for the block must be in incremental mode. The VISTA package insists that the programmers supply incremental input data for the graphical elements within a block. In the IBM package, the programmer may

define a subroutine entity with absolute co-ordinates or incremental co-ordinates. However, he needs two subroutines to achieve this facility. The first specifies the mode and the second defines the graphic element. Consider the representation of a circle by a subroutine entity. Assume the circle of radius 8 raster units is approximated by 72 line segments whose end points are given in arrays IX and IY.

Example 1:

```

        DIMENSION IX(72), IY(72)
        C = 3.141596/180.0
        R = 8.0
        THETA = 5.0
        DO 10 I=1,72
        RADIAN = THETA * I * C
        IX(I) = R * COS(RADIAN)
10      IY(I) = R * SIN(RADIAN)
C      SETS INCREMENTAL MODE
        CALL SDATM (....)
C      GENERATE CIRCLE SUBROUTINE, CORRVAL=2
        CALL BELMT (2,3)
        CALL MVPOS (-, 8,0,-)
        CALL PLINE (-,IX,IY, 72)
        CALL EELMT (2)

```

The IBM package requires a call to SDATM to indicate that the input data IX and IY are in absolute mode. Moreover the programmer has to move the beam to (8,0) which is the beginning point of the first line segment.

In the GRIDSUB, the DLINE is significantly different in that the programmer has the option of defining the block with co-ordinates relative to the existing beam

position (relative mode) or in block co-ordinates (absolute mode) and DLINE also sets the position of the starting point.

Example:

```

        DIMENSION IX(73), IY(73)
        LOGICAL*1 T/.TRUE./, F/.FALSE./
        C = 3.141596/180.0
        R = 8.0
        THETA = 5.0
        DO 10 I = 1,73
        RADIANT = THETA *(I-1)*C
        IX(I) = R * COS(RADIANT)
10      IY(I) = R * SIN(RADIANT)
        CALL BLOCK(2)
        CALL DLINE(F,F,IX,IY,73,F,F,)
        CALL ENDBLK

```

The first two actual arguments F indicate that the first and all other elements in IX,IY are relative to the block origin (absolute mode).

6.2.2 BLOCK and ENDBLK routines

GRIDSUB requires all displayed pictures to be defined as blocks. A block displayed on the screen can be visible or made inactive and light pen detectability on the block can also be changed. However no distinction is made among the blocks according to these attributes as in the IBM package. The controlled entity in the IBM package displays a picture component whose light pen detectability and visibility can be changed and the picture component is always fixed on the screen. A block can have the same properties as a controlled entity if the block is displayed

at the screen position (0,0). Thus the controlled entity is only a particular case of the block. A block can be made invisible with the ERSBK or the ERSID subroutines. The detectability of a block can be set by the LP subroutine.

Nesting of blocks is provided by the ADDBLK subroutine. However a block must be completed before it can be nested in another block. This is parallel to the requirement in the IBM package that a subroutine entity must be closed before it can be nested (by calling LKSUB) in another subroutine entity. The GRIDSUB, however, does not have any provision equivalent to nesting an uncontrolled entity in an uncompleted subroutine entity.

6.2.3 TRANMT

To the programmer this subroutine appears as the EXEC routine in the IBM package. But their functions are quite different. The 2250 is linked to the 1130 through a Storage Access Channel (SAC). The portion of the 1130 core having the 2250 commands representing the image and the logical functions becomes essentially a buffer for the 2250. These commands are accessed by the SAC channel (by cycle stealing) and sent to the display up to 40 times per second. The EXEC only activates the channel. The VISTA package does not contain any subroutine equivalent to TRANMT because the DISPLAY or INSTANT causes the channel to send the

display file to the DD250. Since the GRID is linked to the S/360 with a telecommunication line, the TRANMT initiates input/output routines to send the set of display commands to the GRID which stores the commands in its core and refreshes the display every 20 milliseconds. The TRANMT will wait for messages from the GRID, and when a message is received, it decodes the attention information into a form more natural to the programmer and control is passed to the statement following CALL TRANMT.

6.2.4 Deletion routines

The existing GRID subsystem has only 4K words of core storage which has to accommodate a small supervisor, and input/output routines besides the Display File. Less than 3K words are available for the Display File. The FREEBK subroutine saves space in the GRID core without destroying information stored in the S/360. This feature is unique among the 3 packages. The GRIDSUB is equipped with an internal routine for garbage collection which collects the free space released by FREEBK, DELBLK and DELID subroutines and rearranges the Display File in the GRID core when necessary.

6.3 Evaluation of GRIDSUB

At the outset, the designer has aimed at providing the applications programmer with tools to build a console

command language. As the console user operates with light buttons, that is basic picture components and command words, the design of the package emphasizes on the construction of a picture with basic groups of data which corresponds to the light buttons. The modification of the picture is done through removal or rearrangement of the building blocks rather than modification or extension of the building blocks. This setup is thus most convenient to development of diagrams from basic components, for example, the design of logical circuits, computer flowcharts, or mechanical devices from basic machine parts.

The first version of GRIDSUB lacks formatting facilities. Without provisions for scaling, drawing axes and grid lines, the applications programmer interested in displaying graphs and charts has to provide these facilities himself. However these facilities can be furnished easily as shown in the example in Appendix C.

An application on campus planning is mentioned in Chapter 2. This application typically involves considerable modification of the basic components. As shown in Fig 6-4, the applications program may represent the areas P and Q by two nested blocks of a block. At some stage the console designer may want to combine the regions P and Q as one area.

To change Fig 6-4a to Fig 6-4b, the console designer may point the light pen at the light button 'JOIN', the

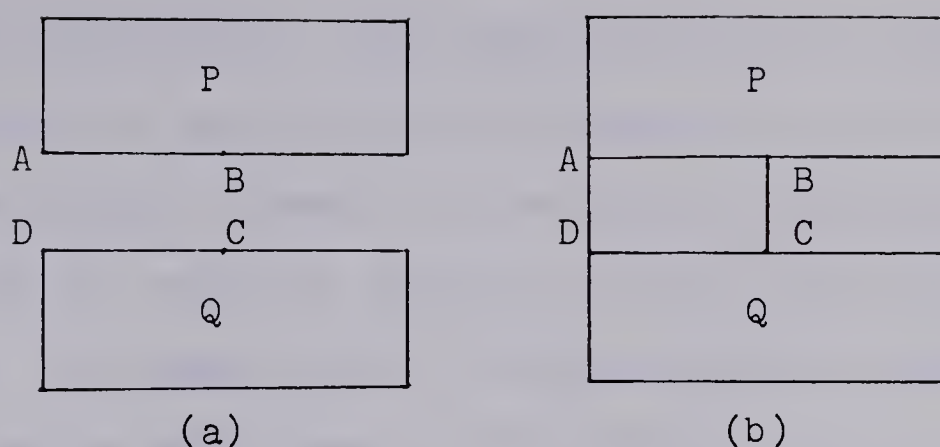


Fig 6-4.

point A, the point D, the light button 'JOIN', the point B, the point C. Lines AB and CD may subsequently be removed with other light buttons. The new set of data should form a new block or an extension of an existing block.

If it is expected that part of a display is to be erased, the contingency may be provided for by defining the part as a nested block or block. Similarly if a certain picture component is expected to be inserted among other picture components, it can be defined as a nested block which is blanked initially by a call to ERSBK. The line segments AB, BC, CD, DA in Fig 6-4 may be defined as nested blocks with AD and BC blanked out initially.

There is one situation in which the nesting facilities are not applicable in the campus planning applications. The applications programmer may not be able to foresee the intersection points of two regions. The consequent manipulation involves handling of graphic elements in picture components rather than relocating, nesting or erasing basic

picture components. This difficulty can be overcome by supplying the applications programmer the relative address of each graphic element in the block during block definition and the relative address in the block when attention arises. By comparing the relative attention address with the relative addresses in block definition, the applications programmer can identify the exact graphic element attentioned.

The nesting facilities of GRIDSUB have an awkward feature. If a block is nested several times in another block, these nested blocks are not distinguishable except by co-ordinates. For example, in a flow-charting problem, a rectangular box with two arrows might be defined as a block with block number 40 as in Fig 6-5.

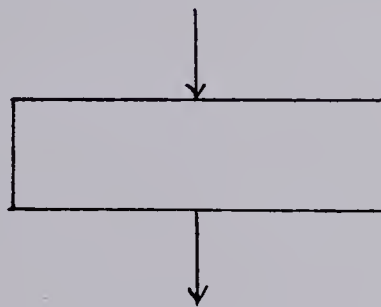


Fig 6-5.

If the two arrows are defined as nested blocks both with number 3, and one of them is picked by the light pen, information obtained in attention is insufficient to reveal which arrow has been light pen attentioned. One possible solution is to define one arrow as block 3, the other as block 4 with block 3 as a nested block. The arrow is still

defined once. Both block 3 and block 4 are nested blocks of block 40.

Several programmers have used the package in writing programs for interactive graphical display. These applications, include logical circuit design, statistical applications and displays of mathematical functions whose parameters and domain are supplied by the console user. The package has been found satisfactory in general. The provision that the programmer can define graphic elements in absolute and relative mode has been found to be particularly convenient. The ADDBLK subroutine was used extensively in the logical circuit design for building the circuit. The message decoding routines immensely lighten the programmer's burden in handling attentions.

CHAPTER VII

IMPLEMENTATION OF GRIDSUB

7.0 Introduction

In general, a FORTRAN applications program using GRIDSUB for graphical display will be composed of the following parts:

(a) Block definition by means of the subroutines BLOCK, ENDBLK and graphic element generation routines.

(b) Calls to DISPLY subroutine

(c) Transmission of DISPLAY FILE to GRID by CALL TRANMT

(d) Attention-processing statements which make the necessary additions, deletions or changes to the display, in response to requests from the on-line operator.

Two types of subroutines are involved in the discussion, the subroutines in GRID core and the subroutines in S/360. We shall refer to the former routines as GRID subroutines and the latter as FORTRAN subroutines or by their names in GRIDSUB. To understand GRID code, refer to Appendix A.

The BLOCK, ENDBLK and graphic element generation routines convert input data into GRID display commands in the BLOCK FILE. The GRID commands are all in incremental mode such that they can be executed repetitively to produce the same display no matter where the starting point is.

Every set of display commands corresponding to a block is copied into the DISPLAY FILE and arranged in the form of a GRID subroutine by the first request to display the block with a call to DISPLY. The GRID subroutines are placed at the 'low' end of the DISPLAY FILE. Each DISPLY call of the block adds to the 'high' end of the DISPLAY FILE a GRID calling sequence to the block. A central free space exists between the blocks and their calling sequences. The hardware label ID is used to identify similar copies of the same block. Thus only one definition of a picture component in the form of a GRID subroutine need be in GRID core even if the picture component is displayed at several places. A calling sequence of seven GRID words is sufficient to establish linkage with the GRID subroutine. This feature helps to reduce the core storage in displaying an item repetitively. Also if a modification is made in the definition of a picture component, all copies of the picture component displayed at various positions will be modified automatically.

Blocks may be nested within other blocks. The definition of a nested block is not included in the block under construction. A calling sequence links the nested block with the outer block. Before a nested block is displayed, an address of the outer block is entered in a HIERARCHY STACK. The state of the STACK reveals the level of nesting and the outer blocks.

Picture components picked by light pen can be identified by the attention address, entries in the HIERARCHY STACK and the GRID hardware label, ID, in the Display State Register. A Display File Table (DFT) records positions of GRID subroutines in GRID core. By comparing the attention address and entries in the HIERARCHY STACK with the DFT, the nested block and outer blocks can be found.

Picture components, nested or independent, can be blanked out by display jump commands to skip the calling sequences to the blocks. The display of blanked picture components can be reactivated by resetting the original contents of the locations where the jump commands were inserted.

Unwanted picture components are destroyed by 'DELETE' calls which blank these components and treat the core space occupied by the corresponding GRID subroutine and calling sequence as free storage space. When the central free space is not enough to accommodate new blocks and their calling sequences, the DISPLAY FILE is compacted such that 'deleted' GRID subroutines and their calling sequences will no longer exist.

Only segments of the modified DISPLAY FILE are transmitted to GRID. Both the number of segments and the length of each segment can increase the transmission time. The maximum number of segments is limited to six in each

transmission.

7.1 The BLOCK FILE and the DISPLAY FILE

For interactive systems, it is likely that what is on the screen at one time is only a small part of the total picture stored in the S/360. There will not be sufficient space to store the definitions of all these blocks in the GRID core. Two files are thus kept in the S/360: The BLOCK FILE to store the definitions of blocks and the DISPLAY FILE which is an image of the GRID core.

Upon completion of the definition of a block, the block will be in the form of a set of GRID display commands in the BLOCK FILE as in Fig 7-1. A specification of GRID commands is given in Appendix A. BLOCK and ENDBLK calls only update the tables. They do not produce any instruction in the BLOCK FILE. Note that all GRID commands are in incremental mode.

When the first call to DISPLY of the block is encountered, the set of GRID display commands is copied into the DISPLAY FILE and arranged in the form of a GRID subroutine by adding jump display (JDD) commands to the beginning and the end of the set of display commands. Each call to DISPLY inserts into the DISPLAY FILE a calling sequence to the appropriate subroutine. This calling sequence supplies the position on the screen for the origin of the picture component represented by the block. Hence for each copy of the

<u>FORTRAN statements</u>	<u>GRID commands*in BLOCK FILE</u>	<u>GRID subroutine in DISPLAY FILE</u>
CALL BLOCK (7)		
CALL POINT	PP2	BLK7 JDD 0
	X ---,RE	
	Y ---	PP2
CALL TEXT	PT2	X ---,RE
	X ---,RE	Y ---
	Y ---	PT2
	DC C'---!'	X ---,RE
CALL DLINE	PP2	Y ---
	X ---,RE	DC C'---!'
	Y ---,BL	PP2
	PV2	X ---,RE
	X ---,RE	Y ---,BL
	Y ---	PV2
	X ---,RE	X ---,RE
	Y ---	Y ---
	.	X ---,RE
	.	Y ---
	X ---,RE	.
	Y ---	.
CALL ENDBLK		X ---,RE
		Y ---
		JDD BLK7

Fig 7-1 A block defined in FORTRAN is converted into a set of GRID commands in BLOCK FILE and a GRID subroutine in DISPLAY FILE.

* See Appendix A for description of GRID commands.

block the calling sequence is shown in Fig 7-2.

<u>FORTTRAN Statement</u>	<u>GRID Calling Sequence</u>
CALL DISPLY (BKNO,ID,X1,Y1)	IDY ID
	PP2
	X X1,AB
	Y Y1,BL
	ELP
	JRD ADBKNO+1

Fig 7-2. A GRID calling sequence generated by a DISPLY call.

Notice that the co-ordinates supplied by the calling sequence for the origin of the block are in absolute co-ordinates. By default, the block displayed is assumed to be detectable by the light pen. ADBKNO is the first address of the GRID subroutine corresponding to the block designated by BKNO in the GRID core. The proper linkage between the GRID calling sequence and the GRID subroutine is also established by the DISPLY subroutine. The calling sequences are placed at the bottom of the DISPLAY FILE i.e. in 'high' core. Each calling sequence is added below (in terms of store address) the preceding one. The DISPLAY FILE is related to the supervisor and input/output routines as shown in Fig 7-3. Blocks from the BLOCK FILE are copied into the top of the DISPLAY FILE, i.e. in 'low' core. The beginning address in the GRID core where the DISPLAY FILE starts is dynamically set by the GRID supervisor. For the

	Supervisor and Input/output routine	
	NESTBK subroutine DEC subroutine	These two GRID subroutines are used to update the HIERARCHY STACK for ADDBLK calls.
3760 ₈	Block 19	First GRID subroutine copied into DISPLAY FILE.
	Block 27	Second GRID subroutine copied into DISPLAY FILE.
	Central Free Space	
7750 ₈	REF	
7751 ₈	GRID calling sequence to Block 27	The second calling sequence issued
	GRID calling sequence to Block 19	The first calling sequence issued
7767 ₈	JDD Ø '7750'	Jump to top of calling sequences
7770 ₈		

Fig 7-3 Relation of DISPLAY FILE to GRID supervisor and Input/output routine in GRID core. Locations 7771₈ to 7777₈ are reserved for I/O transmissions.

purpose of illustration, it is assumed as 3760_8 .
Succeeding blocks copied will follow one another.

Since the DISPLAY FILE is an image of the GRID core which has only 4096 (twelve bit) words, the DISPLAY FILE cannot be longer than 4096 S/360 halfwords. Each halfword has 16 bits of which only the upper 12 bits are used. Since the BLOCK FILE is to accommodate all possible graphical commands, it can be longer than 4096 halfwords.

7.2 Table Handling.

Three tables are kept in the S/360 core: the BLOCK FILE TABLE, DISPLAY FILE TABLE and DISPLAY ID TABLE.

(i) BLOCK FILE TABLE (BFT)

Space is allocated for this table in the form of an integer array BFT(4,100) and two logical vectors BFT2(100) and ERATBL(100) as shown in Fig 7-5. Each element of the integer array takes up a halfword and each element of the logical vectors takes up one byte.

For the initial version, 100 entries are allowed in the BLOCK FILE TABLE. Each nested block is also considered as an entry.

Each entry in BFT has 6 items:

- (a) The number of the block.
- (b) The first address of the block definition in the BLOCK FILE.
- (c) and (d) The X and Y co-ordinate positions in raster

units with respect to the block origin when the definition of the block is closed. This information is necessary in setting the beam position if the block is nested.

For the Ith entry, the above 4 items go into the Ith column of BFT. If the entry corresponds to a nested block, the contents of a,b,c,d are different. A calling sequence ADBKSEQ is inserted within the main block in the BLOCK FILE (see Section 7.4). The block number is turned negative and stored as item (a). Item (b) is the first address of the calling sequence ADBKSEQ in the BLOCK FILE. Item (c) is the address in the main block following this calling sequence and item (d) contains the relative address of the location (LINK+1) in the calling sequence where proper linkage for the nested block is to be set.

- (e) A flag to indicate whether the block has been copied into the DISPLAY FILE so that subsequent DISPLY calls need not copy the block. For the Ith entry, the flag is set at the Ith element of the logical vector BFT2.

If BFT2(I) = .FALSE. the set of GRID commands in

BLOCK FILE has not been

transferred to the DISPLAY FILE.

If BFT2(I) = .TRUE. the block has been transferred

to the DISPLAY FILE.

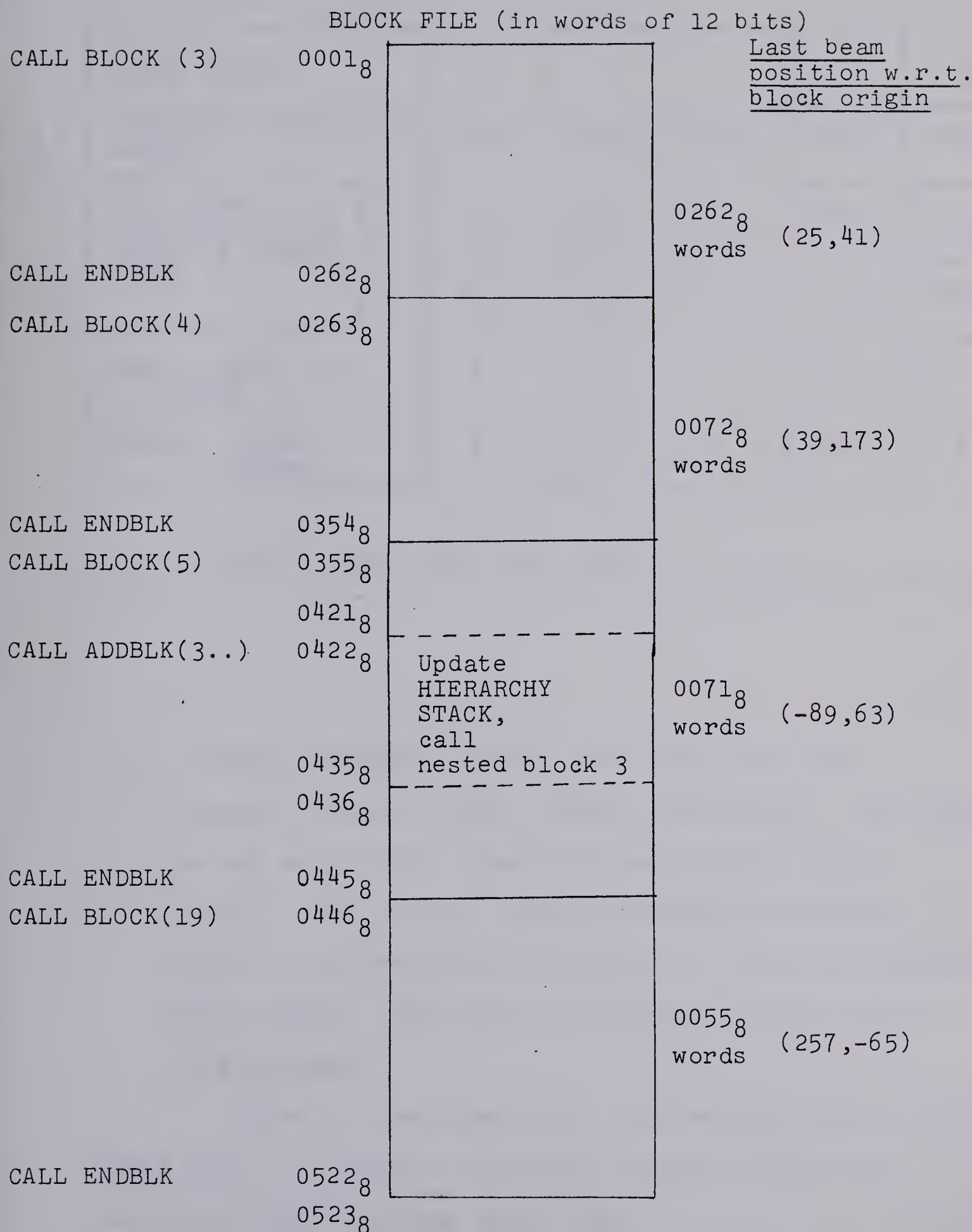


Fig 7-4 The BLOCK FILE not drawn to scale. ICBF points to 0523₈.

B F T	BLOCK NUMBER		3	4	5	-3	19
	BEGINNING BLOCK FILE ADDRESS		0001 ₈	0263 ₈	0355 ₈	0422 ₈	0446 ₈
	LAST BEAM POSITION OF THE BLOCK IN RASTER UNITS	X	25 ₁₀	39 ₁₀	-89 ₁₀	0436 ₈	257 ₁₀
		Y	41 ₁₀	173 ₁₀	63 ₁₀	6 ₁₀	-65 ₁₀
	BFT2 (.TRUE. or .FALSE.)		F	F	F	F	F
	ERATBL (.TRUE. or .FALSE.)		F	F	F	F	F

Fig 7-5 The BLOCK FILE TABLE

(f) A flag to indicate whether the block has been 'erased' through ERSBK, ERSNBK subroutines. The flag is set at the Ith element of the logical vector ERATBL. The internal garbage collection routine GARCOL retrieves this information to set the DISPLAY FILE properly when any rearrangement of the DISPLAY FILE is made.

There is a pointer ICBF (Instruction Counter for BLOCK FILE) to indicate the next available address in the BLOCK FILE for block definition.

For the arrangement of blocks in BLOCK FILE shown

in Fig 7-4, the entries in BLOCK FILE TABLE are shown in Fig 7-5.

When a CALL DISPLY statement is encountered, the BLOCK FILE TABLE is checked to see if the entry in BFT2 corresponding to the block is .FALSE.. If so, the set of GRID commands corresponding to the block will be copied into the DISPLAY FILE. If the entry is .TRUE., the copying is not necessary. In both cases, a GRID calling sequence is added to the DISPLAY FILE

(ii) DISPLAY FILE TABLE (DFT)

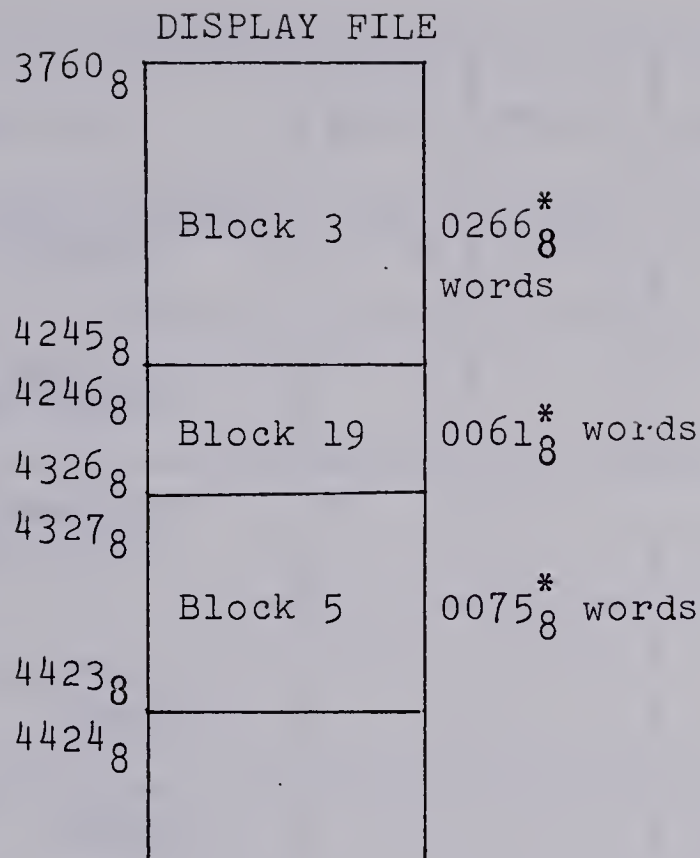
The DISPLAY FILE TABLE is a (2x100) integer array, each element occupying a half word. The relation between an arrangement of blocks in the DISPLAY FILE and the DFT is shown in Fig 7-6.

The first row of the DFT contains the block numbers (negative for nested blocks) and the second row the corresponding beginning address in the DISPLAY FILE. The DISPLAY FILE is restricted to accommodate only 100 blocks and/or nested blocks.

(iii) DISPLAY ID TABLE

The DISPLAY ID TABLE consists of an integer array DSID(4,150) and two logical vectors DSID2 and DSID3, each of dimension 150. Each element of DSID occupies one half word and each element of DSID2, DSID3 takes up only 1 byte. A DISPLAY ID TABLE is shown in Fig 7-7.

Each call to DISPLY generates one BLOCK NUMBER and



*Note that the length of each clock in the DISPLAY FILE is 4 words longer than it was in the BLOCK FILE.

DISPLAY FILE TABLE

BLOCK NUMBER	3	19	5	-3
BEGINNING DISPLAY FILE ADDRESS	3760 ₈	4246 ₈	4327 ₈	4376 ₈ ^{**}

**The first address of the nested (-3) is 47₈ words (compared with 45₈ words in the BLOCK FILE from the beginning of block 5 because a JDD instruction is added at the beginning.

Fig 7-6 The DISPLAY FILE and the
DISPLAY FILE TABLE

DSID	NUMBER -ID	$3*64+1$	$3*64+2$	$19*64+13$	$5*64+1$	$3*64+27$
	BEGINNING DISPLAY FILE ADDRESS	7760_8	7751_8	7742_8	7733_8	7724_8
	X IN ABSOLUTE SCREEN UNITS					
	Y IN ABSOLUTE SCREEN UNITS					
	DSID2 (.TRUE. or .FALSE.)	F	F	F	F	F
	DSID3 (.TRUE. or .FALSE.)	F	F	F	F	F

Fig 7-7 The DISPLAY ID TABLE.

ID combination which forms an entry to the DISPLAY ID TABLE.

The first item in the Ith entry, DSID(1,I), contains the BLOCK NUMBER-ID combination. Since the block number is a positive integer from 1 to 511 and ID ranges from 0 to 63, 9 bits for NUMBER and 6 bits for ID will be sufficient for each pair of NUMBER and ID. The first item contains (NUMBER*64+ID) for each CALL DISPLY (NUMBER,ID, X,Y) eg. 0000001111000011 represents NUMBER=7 and ID=3.

The second item, DSID(2,I), contains the first DISPLAY FILE address of the calling sequence. Since the calling sequences are added below the preceding one,

DSID(2,I) is in descending order for ascending values of I.

The third and fourth items, DSID(3,I) and DSID(4,I), contain the absolute (X,Y) screen co-ordinates where the block is to be displayed.

If the fifth item, DSID2(I), is .TRUE., the copy of the block has been 'erased' through an ERSID call, and if it is .FALSE., the display of the copy of the block is still active.

The sixth item, DSID3(I), controls the light pen detectability. If DSID3(I) = .FALSE., the copy of the block can be detected by light pen.

If DSID3(I) = .TRUE., the copy of the block cannot be detected by the light pen.

7.3 Problem of Relocation in Displaying Picture Components

The GRID subroutine is to display an identical picture component at a screen position specified by the calling sequence. Thus all GRID display commands in the GRID subroutines must be in RELATIVE (incremental) mode. However, a FORTRAN programmer is allowed the freedom to define the elements of a block relative to the last beam position or with respect to the block origin. If an element is defined relative to the last beam position then the equivalent GRID command will be in RELATIVE mode. If an element is defined with respect to the block origin,

the equivalent GRID command will still be in RELATIVE mode. Two variables BEAMX and BEAMY are kept in S/360 during block construction to record the beam position with respect to origin of the block under construction. When ENDBLK is encountered, BEAMX, BEAMY are set to 0.

eg. If BEAMX = 100,
 BEAMY = 70,
 RELBM = .FALSE.
 X4 = 200
 Y4 = 20
 BLINK = .TRUE.
 DASH = .FALSE.

the CALL VECTOR (RELBM, X4,Y4, BLINK, DASH) compares (X4,Y4) with BEAMX, BEAMY to decide the relative increments.

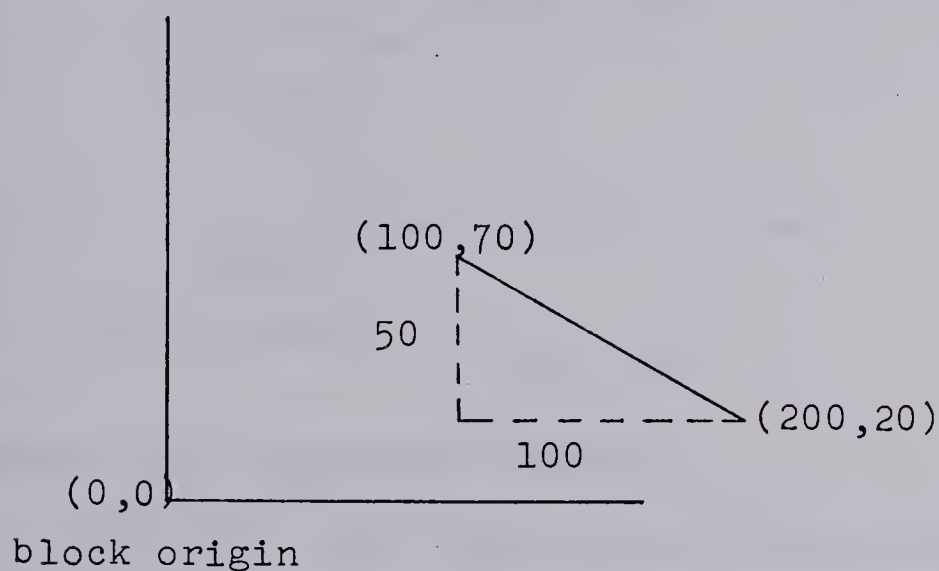


Fig 7-8.

The equivalent GRID command will be

```
PV2
X    100,RE
Y    -50
```

and BEAMX, BEAMY will be set to 200,20 respectively.

If instead, RELBM = .TRUE.

X = 200

Y = 20

and (BEAMX, BEAMY) = (100,70) initially, the equivalent GRID command will be

PV2

X 200,RE

Y 20

and BEAMX, BEAMY will be set to 300,90 respectively.

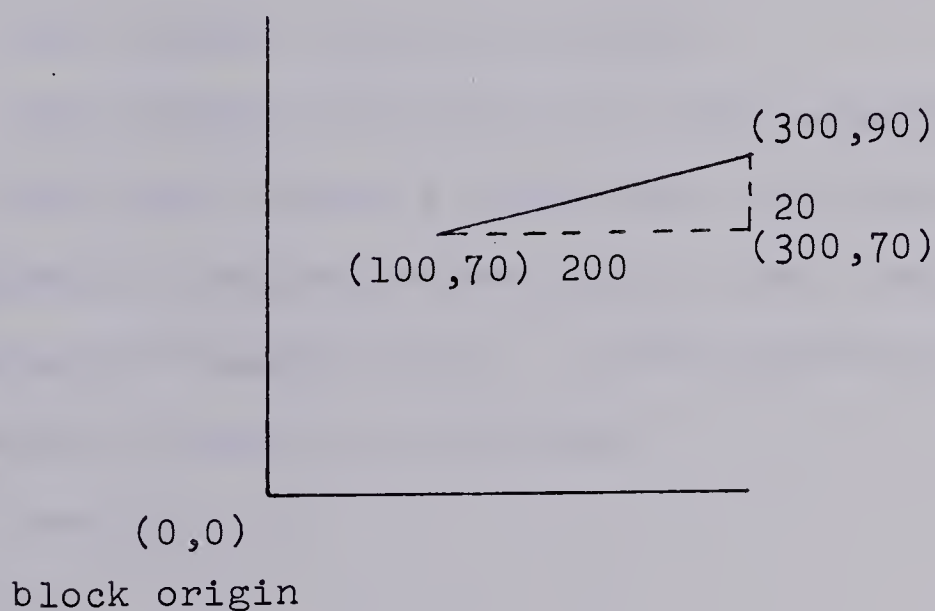


Fig 7-9.

Consider the following examples:

LOGICAL*1 AB/.FALSE./,RE/.TRUE./,F/.FALSE./

For CALL MOVE (AB, 100,70), then the GRID commands

PP2

X 100,RE

Y 70,BL

are generated and

(BEAMX, BEAMY) = (100,70), if initially the beam is at (0,0).

For CALL VECTOR (AB, 200,20,F,F), then the GRID

commands PV2

X 100,RE

Y -50

are generated and

(BEAMX, BEAMY) = (200,20), since the beam is at (100,70) initially.

The strategy to calculate (BEAMX, BEAMY) has to be changed for CALL ADDBLK (NUMBER, RELBM, X,Y) and
CALL TEXT (RELBM, X,Y,N, CHAR,LARGE,BLINK).

Example 1:

```
LOGICAL*1 AB/.FALSE./,RE/.TRUE./
LOGICAL*1 NB/.FALSE./,ND/.FALSE./
CALL BLOCK(5)
```

:

```
CALL VECTOR (AB,200,20,NB,ND)
```

CALL ADDBLK (3,AB,250,120) moves the beam to (250,120) with respect to the origin of block 5, takes the point as the origin for block 3, and displays the commands held under block 3. After displaying block 3, the rest of block 5 is displayed.

(c.f. section 7.4)

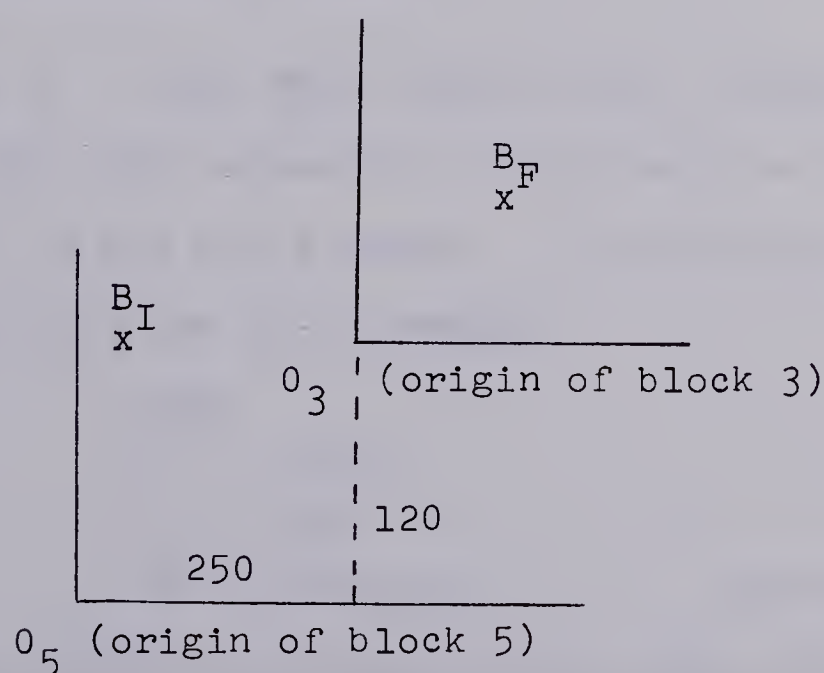


Fig 7-10.

BF = Beam position at the end of displaying nested block 3.

BI = Beam position before displaying the nested block 3.

Since nested blocks can be erased and the instruction following the nested block is in RELATIVE mode, erasing nested blocks may cause anomalies. To allow erasure of the nested blocks without distortion of the main block, the beam, at the end of displaying a nested block, is always moved to the position where the nested block was called.

Therefore (BEAMX, BEAMY) is still (200,20).

In order to move the beam from B_F to B_I , the beam position with respect to block 3 at the end of displaying block 3 must be known. This information is stored in the BLOCK FILE TABLE, BFT. BFT(3,I) contains X and BFT(4,I) contains Y, where the Ith entry in BFT contains information about block 3.

Example 2: CALL TEXT (AB,300,400,4,'ABCD',T,F,)

The first character is displayed at (300,400) with respect to the block origin. If initially (BEAMX, BEAMY) = (230,150), the GRID commands

PT2

X 70,RE

Y 250

DC C'ABCD!' are generated and

(BEAMX, BEAMY) will become $(300 + 16*4, 400) = (364,400)$

7.4 Recognition of Blocks in Light Pen Picks

When the GRID operator picks a displayed block with the light pen, content of the P register is stored in location 0010_8 . This value must be processed so that information regarding block number etc. (see description of TRANMT routine, section 6.1.6) can be passed back to the applications program.

As an example, consider the arrangement of blocks in the DISPLAY FILE as indicated in Fig 7-6. If the light pen in capture mode points at a graphic element of block 3, and attention occurs, the display processor is executing some instruction with address between 3760_8 and 4245_8 . This address which is in the instruction counter, P register, is stored in address 0010_8 and control is passed to the GRID supervisor at location 0011_8 . For block 19, the address returned is between 4246_8 and 4326_8 . Comparison of the contents of location 0010_8 and the second row of the DISPLAY FILE TABLE will give the block number of the block picked.

The GRID instruction repertoire includes an IDY instruction which has the machine code as $54CC$, where CC is an octal number ranging from 00_8 to 77_8 . The command assigns to CC the identification number ID which is an argument in the DISPLY (NUMBER,ID,X,Y) subroutine. This ID is stored in the Display State Register and can be retrieved when attention occurs.

If the light pen points at the nested block 3 of block 5, the address returned will still be between 3760_8 and 4245_8 instead of between 4327 and 4423. There would be, if there were only the knowledge of the interrupt address, no way to tell whether the block interrupted is a nested block or block by itself.

To overcome this problem, a HIERARCHY STACK (a pushdown stack) is maintained in the GRID core. For every ADDBLK call in the construction of a main block (eg. block 5 in Fig 7-4), a set of instructions is inserted in the definition of the main block in the BLOCK FILE for:

- (a) 'pushing down' the HIERARCHY STACK and entering an address of the main block into it
- (b) calling the GRID subroutine corresponding to the nested block, and
- (c) 'popping up' the HIERARCHY STACK.

The HIERARCHY STACK is composed of a pointer STKPT and a table whose first address is LEVEL as shown in Fig 7-11. STKPT is an address $\leq 77_8$, so that one word instructions can be used for indirect addressing. Initially STKPT points at LEVEL-1. Everytime when the execution of display commands gets one level deeper, STKPT is incremented by 1. STKPT is decremented by 1 every time the execution gets one level higher. The table has space for only 7 entries. Hence blocks can only be nested seven levels deep. If, when attention occurs, STKPT still points

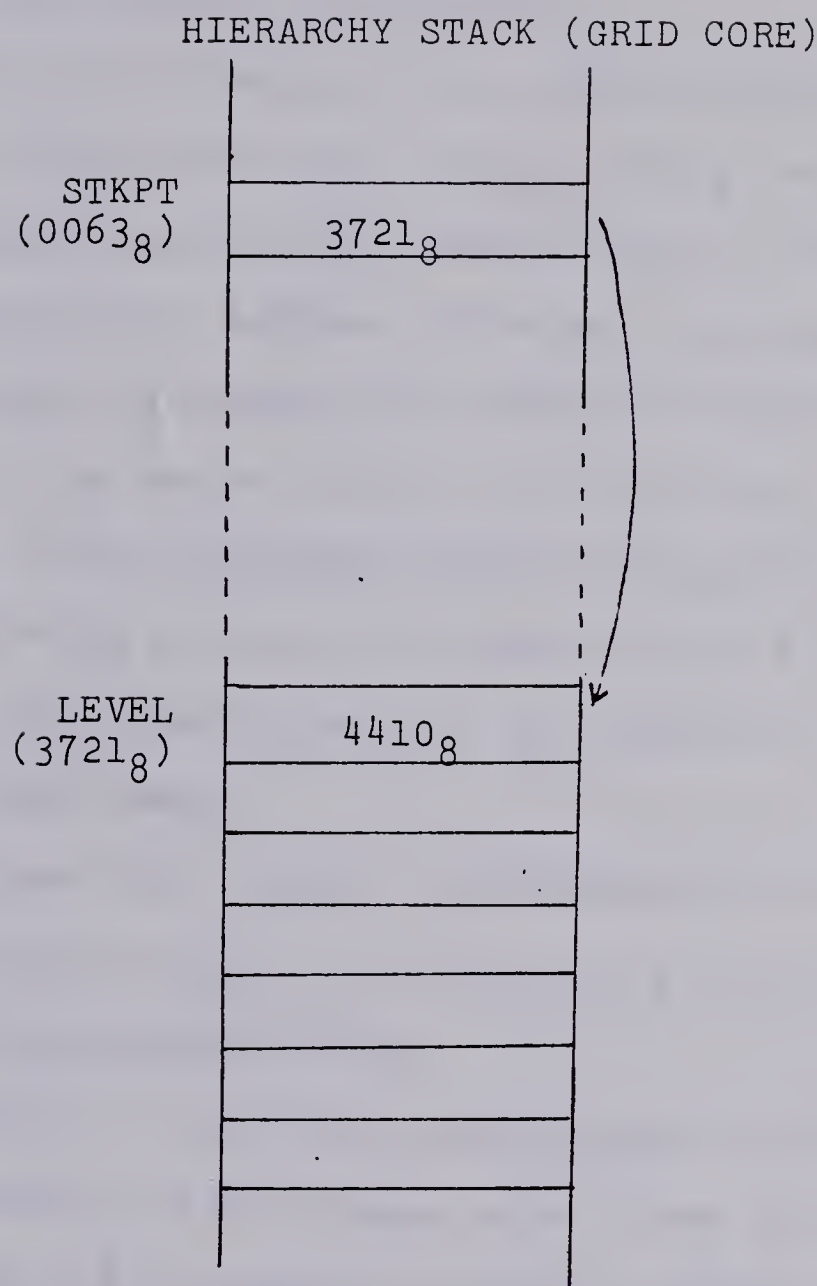


Fig 7-11 The HIERARCHY STACK.

at LEVEL-1, this indicates the block picked or the part of the block picked is not nested.

During the display of the nested block 3, STKPT points at 3721_8 which now contains $4410_8 (=4406_8+2)$. If attention occurs in the nested block 3, the location 0010_8 contains the address ADN where interrupt occurs. By searching the DISPLAY FILE TABLE for ADN the block number of the nested block (3 in this case) can be found. The entry in the HIERARCHY STACK, 4410_8 , which is between 4327_8 and 4423_8 indicates the main block is 5. Content of STKPT, 3721, indicates that the nesting of block 3 is only one level deep.

For each CALL ADDBLK (NUMBER,RELBM,X,Y), the sequence ADBKSEQ shown in Fig 7-12 is generated in the BLOCK FILE to maintain the HIERARCHY STACK.

NESTBK is a GRID subroutine always in GRID core. It puts an address of the higher-level block into the HIERARCHY STACK and increments the stack pointer (STKPT) before it jumps to a nested block. The subroutine DEC decrements the stack pointer STKPT after returning from a nested block. These two stack management routines are shown in Fig 7-13.

The address ADDR in ADBKSEQ is stored in the GRID subroutine NESTBK as return address and also as an entry into the HIERARCHY STACK. BLKN is the first DISPLAY FILE address corresponding to the block designated by NUMBER.

<u>Address</u>	<u>Op Code</u>	<u>Operand</u>	<u>Comment</u>
ADBKSEQ	JRD	NESTBK+1	'PUSHDOWN' HIERARCHY STACK & ENTER ADDRESS OF MAIN BLOCK
ADDR	PP2		
	X	---,RE	
	Y	---,BL	
LINK	JRD	BLKN+1	JUMP TO NESTED BLOCK BLKN
	JRD	DEC+1	
	PP2		
	X	---,RE	RESET THE BEAM POSITION
	Y	---,BL	

Fig 7-12 The GRID subroutine ADBKSEQ to be inserted in the BLOCK FILE for handling nested blocks.

<u>Address</u>	<u>Op Code</u>	<u>Operand</u>	<u>Comment</u>
NESTBK	JPD	0	RETURN ADDRESS PLANTED HERE,
	JDP	NEXT	
X	DC	0	
NEXT	STB	1	STORE X REGISTER IN X,
	AOD	STKPT	PUSHDOWN HIERARCHY STACK,
	LDB	6	ENTER AN ADDRESS OF HIGHER
	STI	STKPT	BLOCK INTO HIERARCHY STACK,
	LDB	5	RESTORE X REGISTER,
	JPB	10	RETURN.
DEC	JPD	0	RETURN ADDRESS PLANTED HERE,
	JDP	NEXT1	
X1	DC	0	
NEXT1	STB	1	STORE X REGISTER IN X1,
	LDD	STKPT	
	SBN	1	POP UP HIERARCHY STACK,
	STD	STKPT	
	LDB	5	RESTORE X REGISTER,
	JPB	10	RETURN.

Fig 7-13 The GRID subroutines NESTBK and DEC.

When this sequence is inserted into the definition of a block, locations NESTBK and DEC can be obtained from some parameters determined by the GRID supervisor, but the location BLKN in DISPLAY FILE cannot be decided during the definition of the block. The content of (LINK+1) is left blank and is set later by the DISPLY subroutine. Since the positioning instruction beginning at ADDR in ADBKSEQ can take up two, three or four words depending upon values of X and Y, ADBKSEQ is variable in length. The relative position of (LINK+1) in ADBKSEQ is stored in the fourth item of the entry corresponding to the nested block in the BLOCK FILE TABLE. For example assume in the construction of block 5 as in Fig 7-4, we have

```
CALL  ADDBLK (3, T, 130, 50)
```

The ADBKSEQ will start at 0422_8 in the BLOCK FILE. Since the X and Y increments require 3 words, the length of ADBKSEQ will be 12. The next address following the sequence ADBKSEQ is $(0422_8 + 14_8)$. The position of (LINK+1) is 6 words from the beginning of ADBKSEQ. This explains the number -3, 0422_8 , 0436_8 , and 6 in the fourth entry of the BFT in Fig 7-5.

If block 5 is to be displayed, it has to be copied from BLOCK FILE to the DISPLAY FILE. The BLOCK FILE TABLE is checked to see if block 5 contains nested blocks (negative block numbers following). If it is found that it has a nested block 3, the BLOCK FILE TABLE will again

be checked to see if block 3 had been transferred to the DISPLAY FILE. If block 3 is not in the DISPLAY FILE, it is copied into it. If block 3 is in the DISPLAY FILE, its first address in the DISPLAY FILE TABLE can be found and BLKN + 1 can be set properly.

7.5 Erasure of Blocks and Nested Blocks

To blank out a block, only a jump instruction (JDD) is necessary to skip over the display commands. For the FORTRAN statement:

```
CALL ERSBK (19),
```

an element by element search is carried out along the first row of the DISPLAY FILE TABLE, shown in Fig 7-6 to get the address 4246 of block 19. Before the erasure, the block may have the following contents:

<u>Address</u>	<u>Op Code</u>	<u>Operand</u>
4246	JDD	0
4247		
4250	PV1	
4251	XY	-, -
4252	XY	-, -
.	.	
.	.	
.	.	
4325	JDD	Ø'4246'
4326		

We need only change the contents of locations 4250

and 4251 to (JDD Ø'4246') thus bypassing the execution of display commands between locations 4252 and 4326.

For RSTBK (19), the BFT has to be searched to obtain the beginning address of block 19 in the BLOCK FILE. According to Fig 7-5, this address is 0446_8 . Contents of locations 0446_8 and 0447_8 of the BLOCK FILE then replace those in locations 4250_8 and 4251_8 in the DISPLAY FILE.

For ERSID and RSTID routines, manipulation is done on the calling sequences at the upper end of the GRID core. The CALL ERSID (19,13) initiates an element by element search in the DISPLAY ID TABLE, Fig 7-7, to obtain the DISPLAY FILE address 7742_8 . Contents of locations 7742_8 and 7743_8 will be replaced by JDD Ø'7751' where location 7751_8 is the beginning of the next calling sequence.

For RSTID (19,13), contents of locations 7742_8 and 7743_8 are set as follows:

7742	IDY	13
7743	PP2	

Nested blocks are erased or reset with a similar procedure. To erase a nested block, eg. block 3 in block 5, both the BLOCK FILE TABLE and the DISPLAY FILE TABLE have to be searched. The DISPLAY FILE TABLE yields the beginning address of ADBKSEQ, which is 4376_8 in Fig 7-6. The BLOCK FILE TABLE in Fig 7-5 indicates the length of ADBKSEQ, which is $(0436_8 - 0422_8) = 0014_8$. A jump display (JDD) instruction replaces the content of 4376_8 and an

octal number $(4376_8 + 0014_8) = 4412_8$ is inserted into 4377_8 so that the execution of ADBKSEQ is skipped.

To unblank an 'erased' nested block 3 in block 5, only a search in the DFT is sufficient. Contents of locations 4376_8 and 4377_8 are changed to:

4376_8	JRD	NESTBK+1
4377_8		

7.6 Garbage Collection

Two attempts with different approaches have been made to manage the space allocation in the GRID core.

In the first attempt, two routines, an 'assigner' and a 'garbage collector' were proposed to be kept in S/360 core. The 'assigner' keeps track of the free space in the DISPLAY FILE. The free storage may be space which has not been used or space that is relinquished after some blocks or calling sequences have been deleted by DELBK, DELID or FREEBK subroutines. When the 'assigner' cannot find an integrated section with adequate storage for a new addition, the 'garbage collector' rearranges the DISPLAY FILE to reclaim as much free space as possible. The operation of the 'assigner' and the 'garbage collector' can be illustrated by Fig 7-14 and Fig 7-15.

The second method of garbage collection is to regenerate the DISPLAY FILE completely when the top of the set of calling sequences meets the bottom of the last

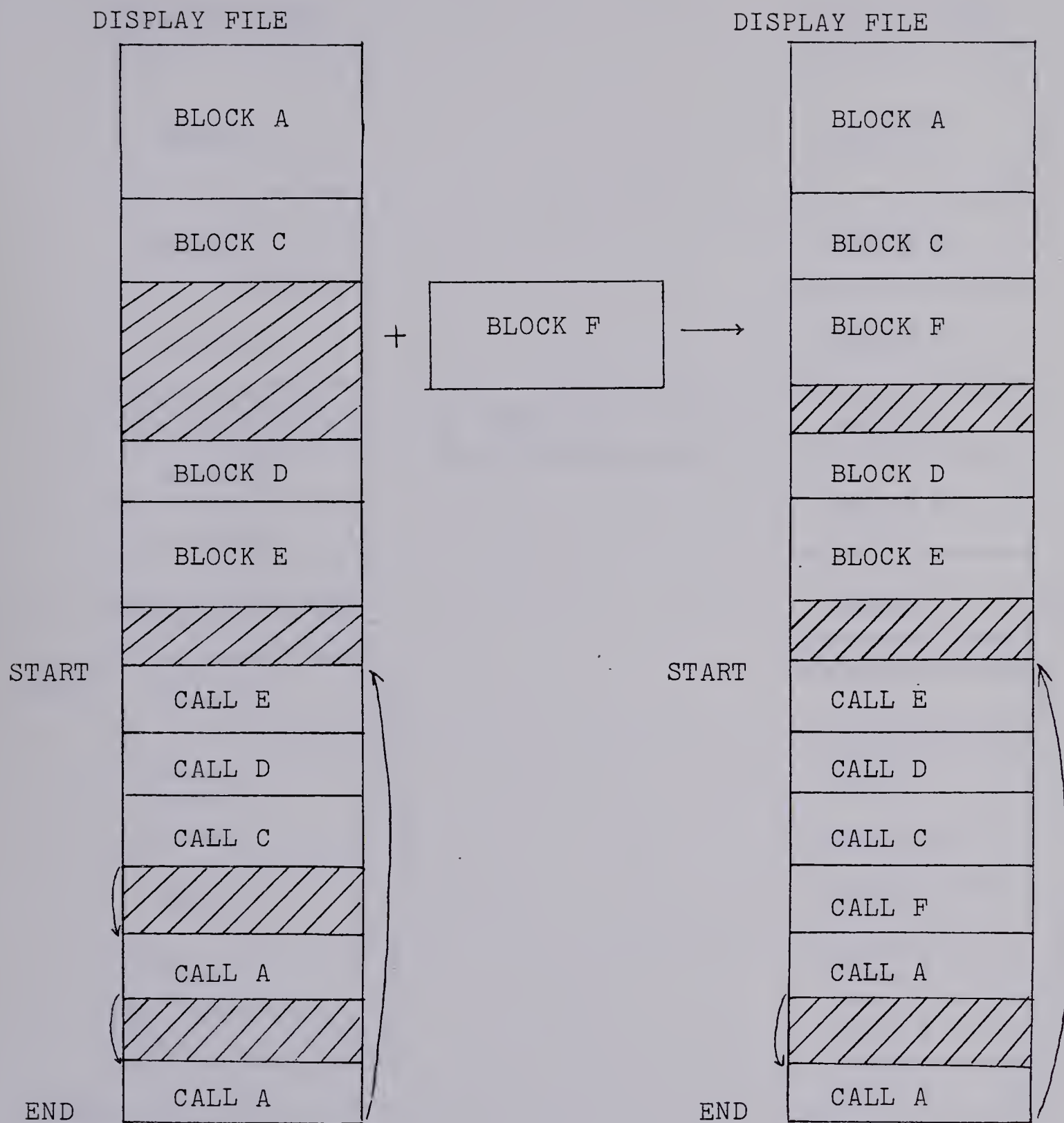


Fig 7-14 The "assigner" assigns new blocks to free storage in the DISPLAY FILE.

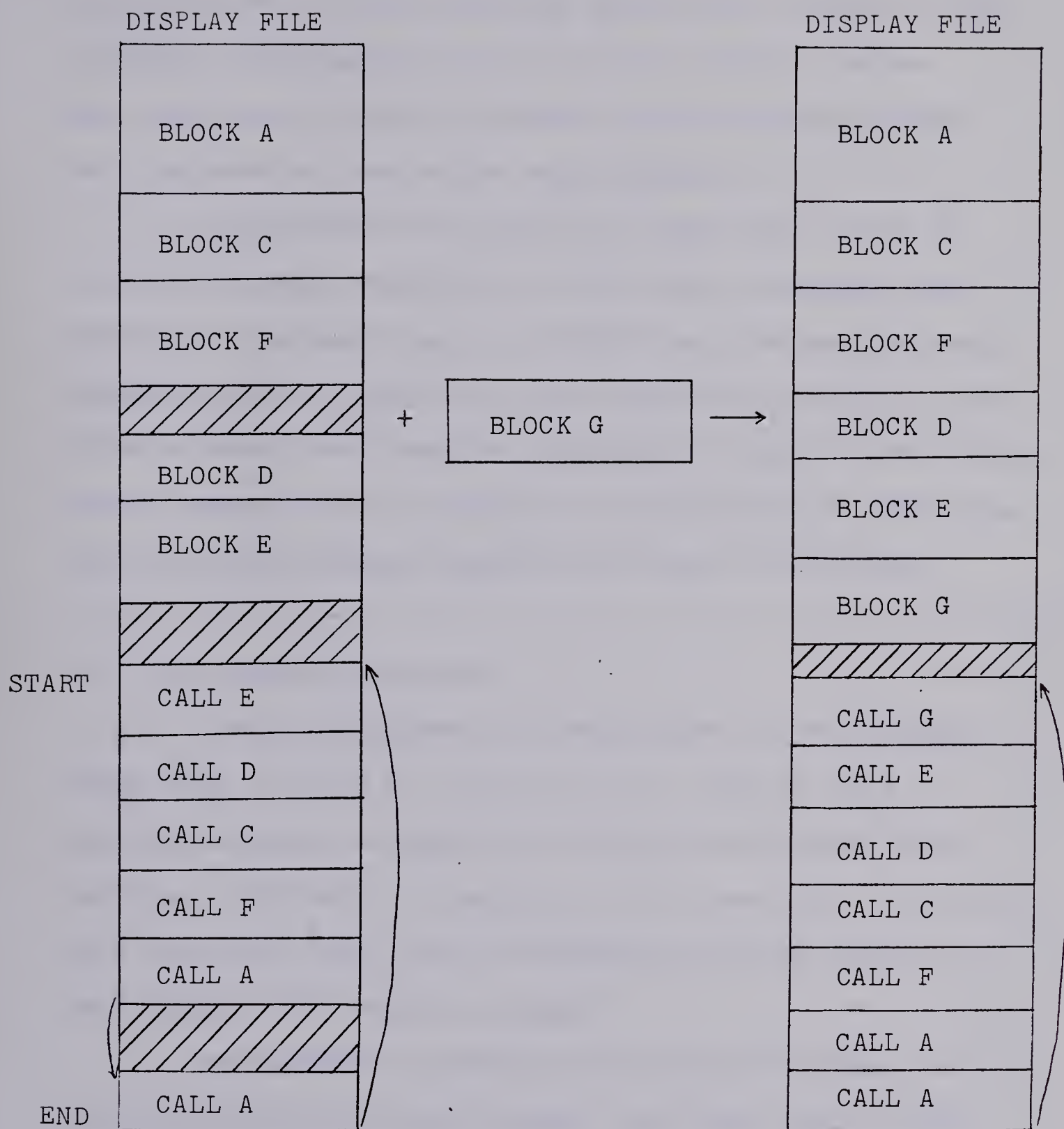


Fig 7-15 The "Garbage Collector" rearranges the DISPLAY FILE.

block entered. Blocks already deleted are omitted. Only a slight modification of the existing DISPLY routine and addition of a short internal routine called GARCOL are required to handle the regeneration.

As the graphical task or at least part of it is to be constantly resident in S/360 core and enjoys top priority in processing, the GRIDSUB and the applications program have to occupy as little space as possible. The first attempt was therefore abandoned in view of the extra table needed in the 'assigner' to keep track of free space and the extra coding required for these two routines.

7.7 The TRANMT Subroutine

A table of pointers is maintained in the package. Every time a block is copied into the DISPLAY FILE, a calling sequence is added or a section of DISPLAY FILE modified, the table is updated so that each set of pointers will show the first and last addresses of the section of the DISPLAY FILE that is changed.

Two hardware oriented input/output routines, one resident in S/360, called GRIDIO, and the other in GRID have been written (K. May 1969). The TRANMT subroutine supplies the table of pointers to GRIDIO which transmits the sections of the DISPLAY FILE indicated by the pointers to the display. The GRIDIO then sets the graphical task in a wait state until such time as the display sends a

message to the S/360. This message is then passed on to TRANMT for interpretation.

As mentioned in Chapter 5, the actual transmission time depends on the overhead of the operation of the transmission routine and the time required to send the information along the telecommunication line. The overhead increases with the number of sections of DISPLAY FILE to be transmitted. A reduction in transmission time by sending the minimum number of display commands in several short segments would incur higher overhead. To achieve a balance between the minimum amount of information transmitted and the minimum number of sections, an internal routine CMPRES is used (F. Jacobsen 1969). The subroutine CMPRES (P, DIF, NO) sorts and compresses the table of pointers.

The parameters of CMPRES are defined as follows:

INTEGER*2 P(2,64)

P(1,I) contains the beginning pointer and P(2,I) contains the end pointer.

NO (INTEGER*2) denotes the number of columns used.

CMPRES first treats each column of the table as a full word and sorts the columns in ascending order, then "compression" is made according to the criterion given by DIF(INTEGER*2) which denotes a 'difference' defined as follows:

If $P(1,I+1) - P(2,I) < DIF$

then $P(2,I) \leftarrow P(2,I+1)$

and $P(1,I+1)$ and $P(2,I+1)$ are eliminated so that

table is 'compressed'. Therefore $NO \leftarrow NO-1$.

It may be possible that

$P(1,I) < P(1,I+1) < P(2,I+1) < P(2,I)$,

then $P(2,I)$ is not changed and $P(1,I+1)$ and $P(2,I+1)$ are eliminated; $NO \leftarrow NO-1$.

The above is also dependent upon

$P(2,I) - P(1,I) < 2000$

i.e. a section of DISPLAY FILE for transmission cannot exceed 2000_{10} words.

Example: Suppose P contains the following values:

400	326	1000	100
440	340	1126	300

After sorting, P becomes

100	326	400	1000
300	340	440	1126

with $NO = 4$

$DIF = 100$

CALL CMPRES(P,DIF,NO)

will modify P as follows

100	1000
440	1126

and set NO to 2.

The optimal balance between the number of sections and the amount of information to be transmitted is still unknown. The first version sets an arbitrary limit of 6 to NO. The 'compressing' is done with the following statements:

```
1      DIF=0
      DIF=DIF+50
      CALL CMPRES(P,DIF,NO)
      IF(NO.GT.6) GO TO 1
      (etc.)
```


CHAPTER VIII

HARDWARE WEAKNESSES IN GRID AND SUGGESTIONS FOR IMPROVEMENT

8.0 Introduction

Several hardware limitations in the GRID have been encountered in the implementation of the package. This chapter discusses the influences these limitations have had on the design of the package. Some suggestions are made for possible improved design of GRID.

8.1 The Labelling Command, IDY

The GRID command repertoire (Appendix A) contains a one-word instruction for providing labelling information on groups of graphical commands. The format is as follows:

<u>Op code</u>	<u>Operand</u>	<u>Machine code</u>
IDY	ID	54cc

where ID is an integer from 0 to 63_{10} , the lower 6 bits of the instruction word, cc, contains the binary representation of ID. This instruction does not affect the display but the value of ID is stored in the lower 6 bits of the Display State Register and can be retrieved when a light pen interrupt occurs. This set of 64 numbers, from 0 to 63) is far from being enough for identification purposes. GRIDSUB resorts to comparing interrupt addresses with tables to recognize blocks, and uses ID for discriminating between calling sequences to the same block.

An attempt was made in the early stages of design of the package to recognize blocks, nested blocks and calling sequences to blocks with labels which are integers from 0 to 4095, giving up completely the hardware facility for labelling. The method is shown as follows: A pushdown stack, PUSHDOWN, which is similar to HIERARCHY STACK in GRIDSUB is required in the GRID core, Fig 8-1. The pointer to the pushdown stack POINTER, is kept in one of the locations between 0 and 77₈. The stack contains 8 entries to generate 7 levels of nesting. POINTER points initially to the address (PUSHDOWN-1), one location before the top of the stack, PUSHDOWN.

The operation of the pushdown stack is shown by the GRID program in Fig 8-2. The block number is stored at BLOCKNO, 7th word of each block in the DISPLAY FILE. The subroutine NAME is used to update the POINTER and enter the name BLOCKNO of the block into the stack. If this approach were adopted, the sequence ADBKSEQ (section 7.4) would call NAME instead of NESTBK. At the end of each block, a call to DECl is made to pop up the pushdown stack.

This approach would give a more uniform labelling method. The applications programmer would be aware of only a label which could be a block number, a block number for a nested block or a label for a calling sequence. This label is similar in appearance to the correlation value in the IBM package. However, the attempt was abandoned in view of

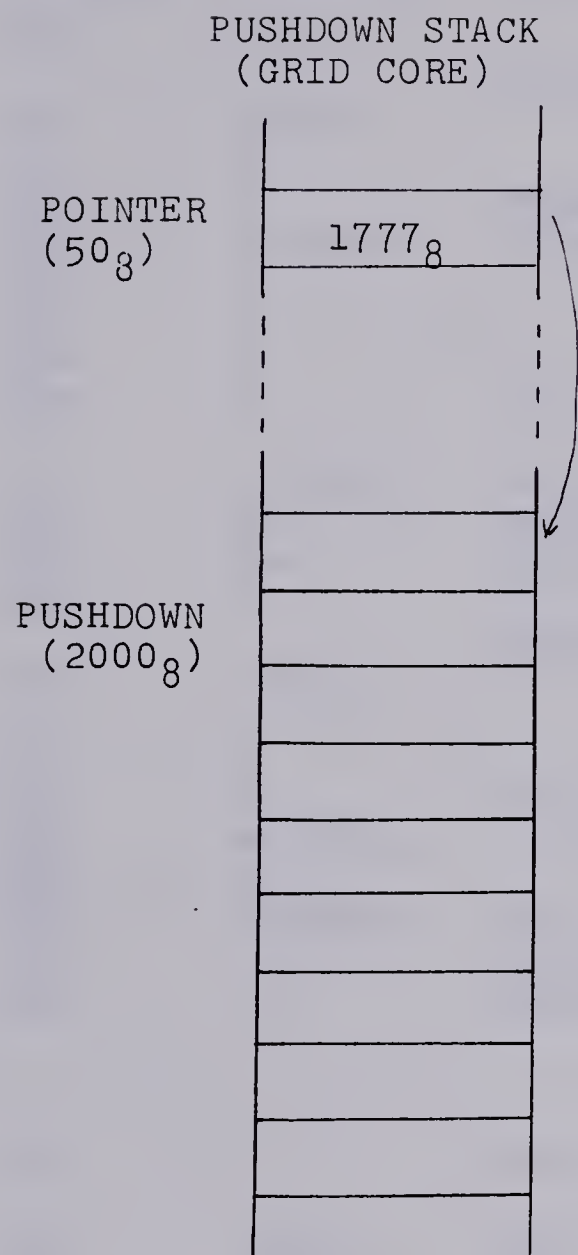


Fig 8-1 - The Pushdown Stack for storing block names.

<u>Address</u>	<u>Op Code</u>	<u>Operand</u>	<u>Comment</u>
NAME	JPD	0	RETURN ADDRESS STORED HERE
	JDP	NEXT3	
X	DC	0	
NEXT3	STB	1	STORE X REGISTER IN LOCATION X
	AOD	POINTER	INCREMENT STACK POINTER
	LDB	6	
	ADN	2	
	STF	2	
	LDM	0	LOAD BLOCKNO
	STI	POINTER	ENTER BLOCKNO IN PUSHDOWN STACK
	LDB	9	RESTORE X REGISTER
	JPB	14	RETURN
DEC1	JPD	0	RETURN ADDRESS STORED HERE
	JDP	NEXT3W	
XX	DC	0	
NEXT3W	STB	1	STORE X REGISTER IN XX
	LDD	POINTER	
	SBN	1	
	STD	POINTER	'POP UP' PUSHDOWN STACK
	LDB	5	RESTORE X REGISTER
	JPB	10	RETURN
	.		
	.		
TOP	JDD	0	BEGINNING OF BLOCK
	JRD	NAME+1	JUMP TO NAME SUBROUTINE
	JDD	FORWARD3	
BLOCKNO	DC	0	
FORWARD3	.		
	.		
	.		
	.		
	.		
	.		
	JRD	DEC1+1	JUMP TO DEC1 SUBROUTINE
END	JDD	TOP	

Fig 8-2 A block with linkages to NAME and DEC1 subroutines which maintain the pushdown stack whose entries are block names.

the extra words (7 words for each block) and the overhead involved. A hardware modification of the IDY instruction would greatly reduce the overhead. Assume IDY is replaced by a two word instruction in the form

<u>F</u>	<u>E</u>	<u>G</u>	<u>Mnemonic</u>	<u>Name</u>
54	00	cccc	LBL	Label

where G, the next word following the instruction 5400, contains a label cccc with 12 bits representing 0 to 4095. If this instruction were provided, all the 12 bits of the Display State Register would be required to accommodate the label stored in G and a new register would have to be built to contain the information at present stored in the upper 6 bits of the Display State Register.

With the LBL instruction, the block definition would be the same as in GRIDSUB. All calling sequences would incorporate the LBL command as IDY.

A pushdown stack would still be required for nested blocks but with the nested block number being entered into the pushdown stack instead of an address in DISPLAY FILE. The GRID subroutines NESTBK1 and DEC1 which are modifications of NESTBK and DEC are shown in Fig 8-3.

This hardware modification would not only provide more uniform labelling of blocks and a wider range of labels but also would save a great deal of table handling. The address of a block in the DISPLAY FILE would still have to be kept for modification purposes. However, entries in

<u>Address</u>	<u>Op Code</u>	<u>Operand</u>	<u>Comment</u>
NESTBK1	JPD	0	RETURN ADDRESS STORED HERE
	JDP	NEXT	
X2	DC	0	
NEXT	STB	1	STORE X REGISTER IN X2
	AOD	STKPT	'PUSHDOWN' HIERARCHY STACK
	STX		STATE REGISTER TO X REGISTER
	STI	STKPT	BLOCK NO. STORED IN STACK
	LDB	5	RESTORE X REGISTER
	JPB	10	RETURN
DEC1	JPD	0	RETURN ADDRESS STORED HERE
	JDP	NEXT3	
X3	DC	0	
NEXT3	STB	1	STORE X REGISTER IN X3
	LDD	STKPT	
	SBN	1	'POP UP' HIERARCHY STACK
	STD	STKPT	
	LDB	5	RESTORE X REGISTER
	JPB	10	RETURN
	.		
	.		
	.		
	.		
	.		
	.		
	.		
ADBKSEQ1	JRD	NESTBK+1	
	LBL		
	DC	----	
	PP2		
	X		
	Y		
	JRD	DEC1+1	

Fig 8-3 The relation between the calling sequence for nested blocks, ADBKSEQ1, and the GRID subroutines NAME and DEC1 if the instructions LBL and STX were implemented.

the pushdown stack would be direct block numbers. No searching in the DISPLAY FILE TABLE would be required to recognize the block where attention has occurred.

8.2 Return Jump Commands.

The GRID instruction set provides two instructions for return jumps, JRD and JPR.

```
Consider      HERE      JRD      ADDR
               HERE      JPR      ADDR
```

Both commands store the address (HERE+2) in the location ADDR and execution starts at ADDR+1. The JRD command is used in display mode and JPR is in processor mode. However, neither can switch mode and at the same time plant a return address in ADDR. It can be seen in the GRID subroutines NESTBK and DEC that two extra words (JDP NEXT) are required to switch operation mode.

<u>Calling Sequence</u>	<u>GRID subroutine</u>
JRD DEC+1	DEC JPD 0
RETURN	JDP NEXT
	X DC 0
	NEXT .
	.
	.
	.
	.

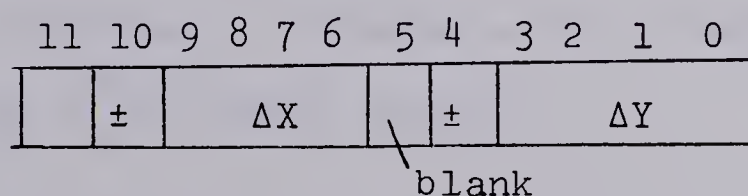
Relocation of the GRID subroutines in GRID core becomes clumsy since NEXT must be an absolute address in GRID core. Addition of return jump commands such as DPR (switching from display mode to processor mode) and PDR

(switching from processor mode to display mode) would be desirable. The DEC subroutine would be simplified as follows:

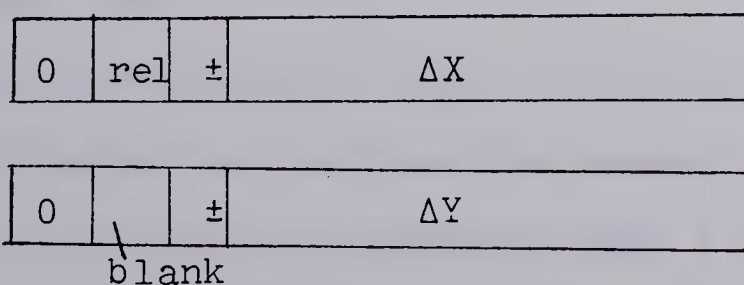
<u>Calling Sequence</u>		<u>GRID Subroutine</u>		
DPR	DEC+1	X	DC	0
		DEC	JPD	0
		STB		3
		LDD	STKPT	
		SBN		1
		STD	STKPT	
		LDB		7
		JPB		7

8.3 Positioning Words

GRIDSUB converts blocks into GRID subroutines in which all position words are in relative mode for reasons explained in section 7.3. Relative positioning can be specified by two modes. Mode I uses only one word with the following format:



Mode II requires two words.



Here the word length in GRID seriously limits the incremental property. Since bit 11 is reserved for dis-

tinguishing instructions from position words, it must be left blank for mode I position words. This arrangement leaves only 4 bits for ΔX or ΔY in mode I besides the sign bits. The increments in mode I words are magnified by one of two scales S1 and S3. S1 doubles the increments and S3 multiplies the increments by 8. Thus mode I instructions can have increments from 2 to 30 in steps of 2 and from 32 to 120 in steps of 8. For this awkward set of increments, considerable programming has to be done to avoid accumulation of errors if Mode I position words are used in graphic element generation routines.

Mode II words allow only 9 bits for X or Y increments. Thus the maximum increment is only 511. Four words are required if the X or Y increment is greater than 511.

It is not possible to allow more addressable increments in the position words as a larger word length such as 16 bits is required. A change in word length would require the redesign of the whole machine.

8.4 The Light Pen

The GRID Supervisor which is used in conjunction with GRIDSUB makes two conflicting demands on the light pen. On one hand, the light pen should have a fairly large field of view for tracking. On the other hand, the light pen must have a small field of view for the 'capture' mode, so that a single character in a string can be picked.

The diameter of the field of view of the light pen on the GRID machine has been found to be about 8 raster units. Even if the tracking cross is drawn with mode II words, only 3 or 4 points along each axis is visible to the light pen. This small aperture renders tracking difficult. Thus the GRID console user can only use a scanning pattern to indicate a blank screen position to the GRID supervisor. However, the fine tip of the light pen facilitates the 'capture' mode of operation. A light pen with adjustable aperture or a set of caps with different apertures which can be attached to the light pen will satisfy the many uses of the light pen without conflict.

CHAPTER IX

SUGGESTIONS FOR VERSION II OF GRIDSUB

9.0 Introduction

Improvements to GRIDSUB are proposed in this chapter. Some are major improvements to make the package more versatile and efficient. Some improvements are minor, being included to augment GRIDSUB for particular uses that might arise.

Examples of major improvements are: recoding GRIDSUB in more efficient code, more flexible error detection, conventions for using secondary storage and choice of GRID supervisor.

A number of subroutines are planned to be included. They are:

MSGBLK which allocates space a block in the BLOCK
FILE for messages to be displayed on the
screen;

DSPMSG which displays a message which uses the
space reserved by MSGBLK;

COPY which copies the definition of a previously
defined block;

DELNBK which destroys a nested block within a block;

EXTBLK which extends the definition of a block;

and a set of routines for scaling and displaying axes or
grid.

9.1 Possible Major Improvements

9.1.1 Recoding the Package

When this project was started, many properties of the graphical display unit were not given in the initial versions of the specification of the machine or were not explicit enough if they were stated. Some characteristics, such as the beam position after a string of characters is displayed with tabular mode, the situation when a line segment extends beyond the screen limits, have been found out only after the display unit was delivered in June 1969. Some GRID commands in the repertoire have even been changed by the manufacturer in the course of the project. Moreover, the package had to interact with the GRID supervisor and transmission routines which were being developed concurrently by other designers. A high level language was deemed necessary for quick program modification and debugging of the package because the design philosophy of software and the familiarity with the hardware were not established. FORTRAN was chosen with the intention of getting a workable package implemented within the shortest period of time. Though an efficient compiler, An IBM FORTRAN H compiler, has been used, the object code generated by the compiler occupies much more space than if the package were written in assembler language. The limit imposed by the size of the available 360 core prevents the

construction of an elaborate package. Equipped with the working experience and confidence in the hardware, the author would implement the package in assembler language if he were to develop the package again.

9.1.2 Error Indication

When errors are detected by the GRIDSUB, such as a request to display a block which has never been defined, an error message is printed and the call is ignored. This is far from being desirable. A modification of all the routines is suggested so that the last argument of each subroutine is an exit for error returns.

Example:

	<u>Applications Program</u>	<u>Subroutine</u>
	.	
	.	
	CALL BLOCK(NUMBER,&99)	SUBROUTINE BLOCK(NUMBER,*)
	.	.
	.	.
	.	.
99	(to handle possible error or stop)	RETURN 1
		.
		.
		.
		RETURN
		END

If BLOCK detects an error, it returns by RETURN 1 and control is passed to the statement labelled 99 in the applications program.

In addition to printing a message to the applications

program, GRIDSUB may use MSGBLK and DSPMSG defined in section 9.2, to set a message on the GRID for the console user before returning by RETURN 1. If this feature is desirable, statement 99 in the applications program should be CALL TRANMT. Though the state of the system may not be of interest to the console user, a simple message such as 'SYSTEM ERROR' when errors like overflow of tables and files may be helpful to the anxious console user who is expecting a reply. He may then terminate the whole job by pressing a function key.

9.1.3 Secondary Storage Facilities

To overcome the limitations imposed by the core size and to satisfy the applications programmer who may define a large number of blocks, the following subroutines could also be implemented in the future:

(a) STRBLK (NUMBER,BF)

Type of variable: INTEGER*4 NUMBER

LOGICAL*1 BF

This subroutine will remove the block designated by NUMBER from core storage to a direct-access file on disk. If a block already exists on disk with the same NUMBER, it will be replaced by the new block transferred from the core storage. The block thus stored on disk may be used at a later time and/or a later date. All the nested blocks of the block are also stored on disk.

Some of the block's nested blocks may be common nested blocks to other blocks or may have been displayed as independent blocks. These nested blocks are not removed from the GRID core, DISPLAY FILE or the BLOCK FILE.

If BF is .FALSE., the block is removed from the GRID core and the DISPLAY FILE but not from the BLOCK FILE, and so are the nested blocks which are only nested in the block. The call is equivalent to storing the block on disk followed by a call to FREEBK.

If BF is .TRUE., the block is removed from the GRID core and all the files and tables in the 360 core and so are the nested blocks which are only nested in the block. The call is equivalent to storing the block on disk followed by a call to DELBLK.

To achieve high efficiency in storage, the blocks stored on disk should be considerably large.

(b) LOADBK (NUMBER,FOUND)

Type of variable: INTEGER*4 NUMBER

LOGICAL*1 FOUND

This subroutine will search the direct-access file for the block identified by NUMBER. If the block is found, it is added to the BLOCK FILE. The applications programmer is responsible to test if the block is copied properly by checking the logical variable, FOUND. If FOUND is true, the block has been transferred to the BLOCK FILE. If

FOUND is false, then the block was not located on the file. The content of the block on disk is not affected by this subroutine.

(c) DELLBK (NUMBER)

Type of variable: INTEGER*4 NUMBER

This subroutine will search the direct-access file and destroy the block designated by NUMBER if it exists. All the nested blocks within this block are deleted.

9.1.4 Choice of GRID Supervisor

The first version of GRIDSUB runs with a GRID supervisor which assembles manual input attentions into a message to be transmitted to the applications program. The GRID supervisor also enables the console user to position picture components on a part of the screen which is blank by displaying a scanning pattern. Future versions might give the console user and applications programmers the option of choosing supervisors to operate in GRID. Two GRID supervisors for line drawing and textual display are being implemented.

9.2 Additions to the Package

With the feedback from the user public beginning to flow in, it is now apparent that some additional features are desirable. The following extensions to the

package are suggested for the second version of GRIDSUB:

(a) A subroutine MSGBLK (NUMBER,COUNT)

Type of variable: INTEGER*4 NUMBER,COUNT

This subroutine would enable the applications programmer to send messages to the console user without defining initially all possible messages in blocks.

The statement, CALL MSGBLK, is equivalent to a call to BLOCK followed with a call to ENDBLK but reserves space in the BLOCK FILE sufficient to accommodate a number of characters given by COUNT.

(b) A subroutine DSPMSG (NUMBER,MESSAG) that displays a block designated by NUMBER which has been defined by MSGBLK.

Type of variable: INTEGER*4 NUMBER

INTEGER*2 MESSAG (array)

The message is supplied by the second parameter, MESSAG, of DSPMSG. The number of characters in the message cannot exceed COUNT, otherwise the extra characters will be ignored. This subroutine allows the applications programmer to define the message when necessary and saves space in the BLOCK FILE. The message is defined as a block so that, after reading the message, the console user may erase the message with manual input devices.

(c) A subroutine COPY (NUMBER)

Type of variable: INTEGER*4 NUMBER

This subroutine copies the contents of a previously defined block named NUMBER and includes it as the whole or a part of the current block under construction. The subroutine permits duplication of previously generated blocks without the necessity either of providing the input data or of duplicating the graphic element generation routines calls used to produce the previous block.

The subroutine provides another method to get over the inconvenience that similar nested blocks in one main block are not recognizable. Consider the same example as in section 6.3. Assume block 3 defines an arrow. The three statements

```
CALL BLOCK (4)
```

```
CALL COPY (3)
```

```
CALL ENDBLK
```

define block 4 with the same contents as block 3. The appearance of block 4 on the screen is the same as that constructed by the following statements

```
CALL BLOCK (4)
```

```
CALL ADDBLK (3,...)
```

```
CALL ENDBLK,
```

but in the first case, block 4 has no nested blocks. As handling of nested blocks takes considerable overhead in the display, it may be advisable to define block 4 with COPY instead of ADDBLK if block 3 is short.

(d) Deletion of nested blocks with DELNBK (NUMBER, NESTBK).

Type of variable: INTEGER*4 NUMBER,NESTBK

This subroutine destroys a nested block within a block. If the nested block designated by NESTBK has other blocks embedded in it, the embedded blocks are also destroyed from the block designated by NUMBER.

The implementation of the above subroutines is already underway. Some other features are still in the planning stage. These include extension of block definitions, EXTBLK, scaling facilities and use of secondary storage.

(e) The subroutine EXTBLK, Extension to Block, is similar to the XELMT in the IBM package. It reopens the definition of a completed block. The extension is defined by graphic element generation routines. The block extended has to be closed by ENDBLK again before it can be displayed.

(f) It is planned to furnish the package with a set of routines that perform scaling, axes and grid displaying. Scaling involves the mapping of the user's plane onto the screen or part of the screen. The plotting of axes and grids is provided for applications in data reduction and statistics. This facility will be similar to AUTO PLOT routines for the Calcomp Plotter.

CHAPTER X

CONCLUSION

It is the author's firm belief that 'A picture is worth a thousand words'. Drawings can bring invaluable understanding to human beings. In Chapter 2, many examples in scientific research have been mentioned in which graphical drawings are indispensable. On-line graphics which gives the console user almost instant response can enhance immensely the user's creativity and working efficiency.

A picture, however, probably requires more than a thousand computer words to construct. Programming in machine codes or assembler languages for non-trivial graphical applications has been found to be tedious and difficult. From the outset of this project, it has always been the author's aim to provide an applications programmer with a package of FORTRAN graphical routines which are easy to learn and simple to use. Equipped with this package, the applications programmer can then easily develop a console command language for the console user who need not be a computer programmer at all. The console user can construct and modify displays via manual input devices. His actions are governed by the console command language which is simple and natural for the application. It is also possible to develop an applications program such that messages are displayed on the screen to guide the console

user's actions.

The VISTA and IBM 2250 packages have been studied and examined critically. The configurations in which they were used were significantly different from that used in this project. However they have served as valuable guides in the design of GRIDSUB, the subroutine package developed in this project. Some of the inconvenient features in the two packages were avoided in the design of GRIDSUB.

The hardware capabilities of GRID were considered. In view of the limited logical and computing power of the display processor, GRID was linked to an IBM 360/67 for non-trivial attention handling and large data bases. For compatibility reasons, a telecommunication link was used instead of a channel to connect the computer and GRID. The medium speed of transmission and the multiprogramming environment ruled out frequent interrupts on the S/360. A GRID supervisor was found necessary to assemble manual input attentions as a message to S/360 instead of routing every attention as an interrupt to S/360. Techniques to achieve faster transmission were also considered.

The specification of GRIDSUB in Chapter 6 may be viewed as a display manual directed towards the graphics applications programmer. Programming considerations and possible error messages are listed. Before the package was released to the user public, it had been tested by several applications programs and found to be satisfactory.

Though some facilities such as scaling and axes-displaying have not been provided they could be coded in the applications program without difficulty.

The internal structure of the package and the interaction among the subroutines have been described in Chapter 7. The methods of converting input graphical data into GRID format were discussed in detail. Emphasis was laid on nesting facilities and table handling techniques in recognizing the picture component attentioned by the light pen and in modifying the display.

The labelling facility provided by the display hardware was found to be insufficient. A slight modification of the GRID labelling instruction, IDY, and addition of a register could greatly simplify programming to recognise picture components attentioned by the light pen. Return jumps are awkward to use and could be changed easily. Incremental XY positioning with one GRID word was not practical. However a modification that involves change in word length was impossible. The light pen was found to be excellent in 'picking' graphical elements but useless for tracking. A light pen with adjustable shutter aperture would be desirable.

Higher efficiency and more versatile services are suggested to be the objectives of the possible second version. Recoding in assembler language, facilities to handle program errors, storage onto disk to overcome core

storage limitations and choice of GRID supervisors have been recommended. A set of subroutines has also been suggested to be added to the package to meet the requirement of some special applications.

REFERENCES

1. Abraham C., 'A Basic Package of Subroutines for using the VISTA DD250 Visual Display Unit', Computer Research Section Technical Report No. 19, C.S.I.R.O., Oct. 1966.
2. Allen T.R. and Foote J.E., 'Input/Output Software Capability for a Man Machine Communication and Image Processing System', AFIPS Conf. Proc., FJCC 26, pp. 387-396, 1964.
3. Britt P.M., Dixon W.J. and Jennrich R.I., 'Time-sharing and Interactive Statistics', UCLA.
4. CDC, 'GRID Engineering Specification', Data Display Division, CDC, 1968.
5. Chasen S.H. (1), 'The Introduction of Man Computer Graphics into Aerospace Industry', AFIPS Conf. Proc., FJCC 27, pp. 883-891, 1965
6. Chasen S.H. (2), 'APT-less Contouring Tapes?' American Machinest, vol. 109, pp. 69-70, July 5, 1965.
7. Coons S.A., 'An Outline of the Requirements of a Computer Aided Design System', AFIPS Conf. Proc., SJCC 23, pp. 299-304, 1963.
8. Cross P., 'A Logical Drawing Board for the PDP7/340', Computer Bulletin, pp. 237-245, Dec. 1967.

9. Deecker, G.F., 'A Comparison of Methods for Digital Encoding of Arbitrary Line Figures', University of Alberta Computing Review, vol. 2, pp. 103-118, March 1969.
10. DEC, 'Programmed Buffered Display Type 338', Digital Equipment Corporation.
11. Dertouzos M.L. (1), 'CIRCAL, On-line circuit Design', Proc. IEEE, vol. 55 No. 5, pp. 637-654, May 1967.
12. Dertouzos M.L.(2), 'An Introduction to On-line Circuit Design', Proc. IEEE, Vol. 55, No. 11, pp. 1961-1970, Nov. 1967.
13. Eisenbies J.L., 'Conventions for Digital Data Communication Link Design', IBM Systems Journal, Vol. 6, No. 4, 1967.
14. Freeman H. (1), 'On the Encoding of Arbitrary Geometric Configurations', IRE (E.C. 10), June 1961
15. Freeman H. (2), 'Techniques for the Digital Computer Analysis of Chain Encoded Arbitrary Plane Curves', Proc. Nat. Electronics Conf., vol. 17, 1961.
16. Gurley B.M. and Woodward C.E., 'Light Pen Links Computer to Operator', Electronics, Vol. 32, pp. 85-87, Nov. 1959.

17. Hamming R.W., 'Numerical Methods for Scientists and Engineers', McGraw Hill Co., 1962.
18. Horwood E.M., 'Computer Applications to Urban Planning and Analysis and Prospects', Proc. IFIP Congress 1968.
19. Hough M., 'Developing a Sense of Place for Community Colleges', Canadian University, Jan-Feb. 1968.
20. IBM Form C27-6934-0, '1130/2250 Graphic Subroutine Package', Systems Reference Library, 1967.
21. IBM Form C28-6535-1, 'IBM System/360 Operating System -- Concepts and Facilities', Systems Reference Library, 1967.
22. IBM Form Y28-6659-2, 'IBM System/360 Operating System --MVT Supervisor Program Logic Manual', Systems Reference Library, 1968.
23. Jacks E.I., 'A Laboratory for the Study of Graphical Man-Machine Communication', AFIPS Conf. Proc., FJCC 26, pp. 363-386, 1964.
24. Jacobsen F., 'GRID Subroutine CMPRES', Internal Report, Dept. of Computing Science, University of Alberta, June 1969.

25. Johnson B.V., 'Preliminaries to a Computer Aided Logic Design System', University of Alberta Computing Review, Vol. 2, pp. 80-102, March 1969.
26. Knowlton K.C., 'A Computer Technique for Producing Animated Movies', AFIPS Conf. Proc., SJCC 25, pp. 57-87, 1964.
27. Kuo F.F., Magnuson W.G. and Walsh W.J., 'Computer Graphics in Electronic Design', Datamation, pp. 71-79, March 1969.
28. Licklider J.C.R. and Clark W.E., 'On-line Man Computer Communication', AFIPS Conf. Proc., SJCC 21, pp. 113-128, 1962.
29. May K., 'GRID Subroutines, /360 to GRID, GRID to /360 Transmission Routines', Internal Report, Dept. of Computing Science, University of Alberta, April 1969.
30. May K. (2) 'GRID I Assembler Language', Internal Report, Department of Computing Science, University of Alberta, 1969.
31. Morse S.P., 'Computer Storage of Contour Map Data', Proc. 23rd National Conf., ACM, pp. 45-51, 1968.

32. Newman W.M., 'An Experimental Program for Architectural Design', Computer Journal, Vol. 9, pp. 21-26, 1966.
33. Ninke N.H., 'Graphic I--A Remote Graphical Display Console System', AFIPS Conf. Proc., FJCC 27, pp 839-846, 1965.
34. Pfaltz J.A. and Rosenfeld A., 'Computer Representation of Planar Regions by their Skeletons', Comm. ACM, Vol. 10, No. 2, 1967.
35. Prince M.D., 'Man-Computer Graphics for Computer-Aided Design', Proc. IEEE, Vol. 54, No. 12, pp. 1698-1708, 1966.
36. Romney G.W., 'Real Time Display of Computer Generated Half Tone Perspective Pictures', IFIP Congress 1968.
37. Rosenfeld A. and Pfaltz J.L., 'Sequential Operations in Digital Picture Processing', Journal ACM, Vol. 13, No. 4, Oct. 1966.
38. Stotz R., 'Man-Machine Console Facilities for Computer Aided Design', AFIPS Conf. Proc., FJCC 23, pp. 323-328, 1963.
39. Sutherland I.E., 'SKTECHPAD: A Man-Machine Graphical

Communication System', Lincoln Laboratory,
M.I.T., 1963.

40. Vorhaus A.H., 'General Purpose Display System',
Datamation, pp. 59-64, July 1966.
41. Wagner F.V. and LaHood J., 'Computer Graphics-
Software Design', Computer Graphics, edited by
F. Gruenberger, Thompson Book Co., 1967.
42. Wallington C.E., 'Display Systems in Numerical
Meteorological Experiments', Australian Computer
Journal, Vol. 1, No. 3, Nov. 1968.
43. Zajac E.E., 'Simulation of a Two Gyro Gravity Gradient
Attitude Control System', Bell Telephone Labora-
tories, (16mm. black and white film, 4 min., sound).

•

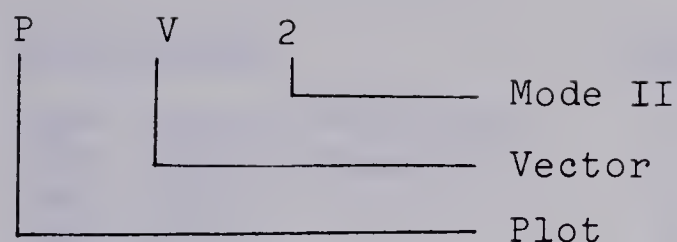
-

•

•

•





The complete set of commands is set as follows:

<u>MNEMONIC</u>	<u>DESCRIPTION</u>	<u>TYPE</u>	<u>OPERAND</u>
ACJ	BANK CONTROL	K	YES
ADB	ADD BACKWARD	B	"
ADC	ADD CONSTANT	C	"
ADD	ADD DIRECT	D	"
ADF	ADD FORWARD	F	"
ADI	ADD INDIRECT	I	"
ADM	ADD MEMORY	M	"
ADN	ADD NO ADDRESS	N	"
AOB	REPLACE ADD 1	B	"
AOC	"	C	"
AOD	"	D	"
AOF	"	F	"
AOI	"	I	"
AOM	"	M	"
BTX	BANK CONTROL TO X	K	NO
CIL	CLEAR INT. LOCKOUT	Z	"
DAD	DISABLE ALL DEVICES	Z	"
DC	DEFINE CONSTANT	ASSEMBLER	YES
DKB	DISABLE KEYBOARD	Z	NO
DLP	DISABLE L.P.	Z	"
DRJ	BANK CONTROL	K	YES
DS	DEFINE STORAGE	ASSEMBLER	YES
EAD	ENABLE ALL DEVICES	Z	NO
EKB	ENABLE KEYBOARD	Z	"
ELP	ENABLE L.P.	Z	"
END	END	ASSEMBLER	YES
ERR	ERROR STOP	Z	NO
EXC	EXT. FUNCTION	C	YES
EXF	EXT. FUNCTION	F	"
HLT	HALT	N	"
IDS	INT. DATA SOURCE	Z	NO
IDY	IDENTIFIER	N	YES
INP	NORMAL INPUT	FM	YES
INX	INPUT TO X	X	NO
IRJ	BANK CONTROL	K	YES
JDD	JUMP DISPLAY-DISPLAY	M	"

<u>MNEMONIC</u>	<u>DESCRIPTION</u>	<u>TYPE</u>	<u>OPERAND</u>
JDP	JUMP DISPLAY-PROCESSOR	M	YES
JFI	JUMP FORWARD INDIRECT	F	"
JMB	JUMP MINUS	B	"
JMF	JUMP MINUS	F	"
JNB	JUMP NOT ZERO	B	"
JNF	JUMP NOT ZERO	F	"
JPB	JUMP POSITIVE	B	"
JPD	JUMP PROCESSOR-DISPLAY	M	"
JPF	JUMP POSITIVE	F	"
JPI	JUMP INDIRECT	I	"
JPR	JUMP RETURN	M	"
JRD	JUMP RETURN DISPLAY	M	"
JZB	JUMP ZERO	B	"
JZF	JUMP ZERO	F	"
LCB	LOAD COMPLEMENT	B	"
LCC	LOAD COMPLEMENT	C	"
LCD	LOAD COMPLEMENT	D	"
LCF	LOAD COMPLEMENT	F	"
LCI	LOAD COMPLEMENT	I	"
LCM	LOAD COMPLEMENT	M	"
LCN	LOAD COMPLEMENT	N	"
LDB	LOAD	B	"
LDC	LOAD	C	"
LDD	LOAD	D	"
LDF	LOAD	F	"
LDI	LOAD	I	"
LDM	LOAD	M	"
LDN	LOAD	N	"
LPB	LOGICAL PRODUCT	B	"
LPC	LOGICAL PRODUCT	C	"
LPD	LOGICAL PRODUCT	D	"
LPF	LOGICAL PRODUCT	F	"
LPI	LOGICAL PRODUCT	I	"
LPM	LOGICAL PRODUCT	M	"
LPN	LOGICAL PRODUCT	N	"
LS1	LEFT SHIFT 1	Z	NO
LS3	LEFT SHIFT 3	Z	"
LS6	LEFT SHIFT 6	Z	"
MUT	MULTIPLY BY 10	Z	"
NOP	NO-OPERATION	Z	"
ORG	DEFINE ORIGIN	ASSEMBLER	YES
OTX	OUTPUT FROM X	Z	NO
OUT	NORMAL OUTPUT	FM	YES
PP1	PLOT POINT TYPE 1	P	OPTIONAL
PP2	PLOT POINT TYPE 2	P	"
PS1	PLOT SYMBOL TYPE 1	P	"
PS2	PLOT SYMBOL TYPE 2	P	"
PTX	P-REG TO X	Z	NO
PT1	PLOT TEXT TYPE 1	P	OPTIONAL

<u>MNEMONIC</u>	<u>DESCRIPTION</u>	<u>TYPE</u>	<u>OPERAND</u>
PT2	PLOT TEXT TYPE 2	P	OPTIONAL
PV1	PLOT VECTOR TYPE 1	P	"
PV2	PLOT VECTOR TYPE 2	P	"
RAB	REPLACE ADD	B	YES
RAC	REPLACE ADD	C	"
RAD	REPLACE ADD	D	"
RAF	REPLACE ADD	F	"
RAI	REPLACE ADD	I	"
RAM	REPLACE ADD	M	"
REF	REFRESH	Z	NO
RMX	R2 REG TO X	X	"
RSX	STATUS REG TO X	X	"
RTX	R1 REG TO X	X	"
SBB	SUBTRACT	B	YES
SBC	SUBTRACT	C	"
SBD	SUBTRACT	D	"
SBF	SUBTRACT	F	"
SBI	SUBTRACT	I	"
SBM	SUBTRACT	M	"
SBN	SUBTRACT	N	"
SCB	SELECTIVE COMPLEMENT	B	"
SCC	SELECTIVE COMPLEMENT	C	"
SCD	SELECTIVE COMPLEMENT	D	"
SCF	SELECTIVE COMPLEMENT	F	"
SCI	SELECTIVE COMPLEMENT	I	"
SCM	SELECTIVE COMPLEMENT	M	"
SCN	SELECTIVE COMPLEMENT	N	"
SDC	BANK CONTROL	K	"
SIC	BANK CONTROL	K	"
SIL	SET INTERRUPT LOCK	Z	NO
SRB	SHIFT REPLACE	B	YES
SRC	SHIFT REPLACE	C	"
SRD	SHIFT REPLACE	D	"
SEF	SHIFT REPLACE	F	"
SRI	SHIFT REPLACE	I	"
SRJ	BANK CONTROL	K	"
SRM	SHIFT REPLACE	M	"
STB	STORE	B	"
STC	STORE	C	"
STD	STORE	D	"
STF	STORE	F	"
STI	STORE	I	"
STM	STORE	M	"
STX	STATE REG TO X	X	NO
X	TYPE 2 POSN WORD	ASSEMBLER	YES
XMA	X REG TO A2	X	NO
XTA	X REG TO A1	X	"
XTR	X REG TO CONTLREG	X	"
XTY	X REG TO Y	X	NO

<u>MNEMONIC</u>	<u>DESCRIPTION</u>	<u>TYPE</u>	<u>OPERAND</u>
XY	TYPE 1 POSN WORD	ASSEMBLER	YES
Y	TYPE 2 POSN WORD	ASSEMBLER	YES
YTX	Y REG TO X	X	NO

Some of the GRID commands that are used in Chapter 7 are explained in the following paragraphs.

(i) Type Plot has two modes. Mode I commands require one position word which contains the X and Y increments. Mode II requires two position words to hold X and Y co-ordinates (if a parameter of X position word is ABsolute) or X and Y increments (if a parameter of X position word is RElative).

	<u>Op Code</u>	<u>Operand</u>
Example 1:	PV2	
	X	-100,RE
	Y	60
	X	30,AB
	Y	40

Two vectors are drawn. The first vector is drawn from initial position (X_i, Y_i) to $(X_i - 100, Y_i + 60)$, from which the second vector is drawn to absolute screen position $(30, 40)$.

Example 2:	PP1	
	XY	20,30,BL

The beam is moved from initial position (X_i, Y_i) to $(X_i + 20, Y_i + 30)$. The point is not displayed (BLank).

Example 3:

PT2	VT, LG
X	150, RE
Y	75
DC	'ABCD!'

Assume the initial beam position is (X_i, Y_i) . The vertical (VT) string of large (LG) characters ABCD is displayed beginning with the centre of A at (X_i+150, Y_i+75) . The character ! which is an 'escape' character indicating the end of string is not displayed.

(ii) Direct Address Type (D): The operand denotes one of the first 100_8 locations of the GRID core.

Example:

<u>Symbolic Address</u>	<u>Location</u>	<u>Op Code</u>	<u>Operand</u>
STKPT	0063_8	DC	3000
	*	*	*
	5721_8	LDD	STKPT
	*	*	*
	6215_8	AOD	STKPT

LDD loads the contents of STKPT into the X register.

AOD loads contents of STKPT into the X register, increments the X register by 1, stores contents of X register in STKPT.

(iii) No Address Type (N): The operand, whether it be

absolute or symbolic, must be a constant between 0 and 77_8 .

Example:	<u>Location</u>	<u>Op Code</u>	<u>Operand</u>
		SBN	1

1 is subtracted from the contents of the X register.

(iv) Memory Address Type (M): This command occupies two words of core storage, the second of which contains the address of the operand.

Example:

<u>Symbolic Address</u>	<u>Location</u>	<u>Op Code</u>	<u>Operand</u>
	5621_8	JDP	NEXT
	5622_8		
X	5623_8	DC	0
NEXT	5624_8		

The JDP command transfers control to the address given by the operand, NEXT, which is 5624_8 .

(v) Indirect Address Type (I): The operand references 77_8 of the first 100_8 locations. The contents of the operand becomes the operand address.

Example:

<u>Symbolic Address</u>	<u>Location</u>	<u>Op Code</u>	<u>Operand</u>
STKPT	0063_8	DC	'Ø'3001'
	*	*	*

<u>Symbolic Address</u>	<u>Location</u>	<u>Op Code</u>	<u>Operand</u>
	6757 ₈	STI	STKPT

The content of the X register is stored in the address as 3001 octal which is specified by the contents of STKPT.

(vi) Relative Backward Address (B): The operand which is between 0 to 77₈ is subtracted from the contents of P register to form an effective address.

Example:	<u>Location</u>	<u>Op Code</u>	<u>Operand</u>
	5321 ₈	LDB	5

The content of location (5321₈-5 = 5314₈) is loaded into the X register.

(vii) Register to Register Type (X): The content of one register is transferred to another.

Example:	<u>Location</u>	<u>Op Code</u>	<u>Operand</u>
		STX	

The content of the Display State Register is transferred to the X register.

APPENDIX B

FORMAT OF MSGTBL AND MESSAG

The message returned from GRID to 360 is assembled into two arrays: MSGTBL and MESSAG. The FORTRAN programmer can either process the information in these arrays or use the decoding routines described in section 6.1.6. An outline of a program using the decoding routines appears in Appendix D. The contents of MSGTBL and MESSAG are described here.

```
INTEGER*2  MSGTBL(3,20),  MESSAG(150)
```

```
COMMON / MESSAGE / MSGTBL, MESSAG
```

MSGTBL is a table whose 20 entries correspond, in order, to the (up to) 20 components in the GRID console user's message.

MSGTBL (1,I) specifies the component type for the Ith component:

- =1 for light pen
- =2 for status/function
- =3 for a string of alphanumeric characters
- =4 for a string of connected vectors
- =5 for a string of points
- =0 for "end of message".

MSGTBL (2,I) specifies the subscript value for the first of a series of elements in MESSAG for the Ith component.

MESSAG (n_1+1) - integer value (0-9) of function key value or

MESSAG (n_1+1) = -1 if the interrupt key, rather than a function key, has been hit.

3. Alphanumeric String

MSGTBL (1,I) = 3

MSGTBL (2,I) = n_1 (say)

MSGTBL (3,I) = n_2 (say)

MESSAG (n_1), MESSAG (n_1+1) = IX, IY, co-ordinates of the center of the first symbol of the string.

MESSAG (n_1+2).....MESSAG (n_1+n_2-1) each contains two characters of the string, in EBCDIC and left-to-right. If the number of the characters is odd, the rightmost character of the last word is left blank. (n.b.: $n_2-2 \leq 43$ - max. of 86 characters in a line)

4. Vector String

MSGTBL (1,I) = 4

MSGTBL (2,I) = n_1 (say)

MSGTBL (3,I) = n_2 (say)

MESSAG (n_1) to MESSAG (n_1+n_2-1) - an even number of words - giving in pairs, the co-ordinates of the vector end points. For example; if

MSGTBL (3,I) specifies the number of elements.

1. Light Pen:

$$\text{MSGTBL}(1, I) = 1$$
$$\text{MSGTBL } (2, I) = n_7 \text{ (say)}$$
$$\text{MSGTBL } (3, I) = n_2 \text{ (say)}$$

Then,

MESSAG (n₁), MESSAG (n₁+1) - IX, IY, the
co-ordinates of the point, center of symbol, or end of the
vector picked with the pen.

MESSAG (n₁+2) = 0 if a point has been picked
 = 1 if a symbol has been picked
 = 2 if a vector has been picked

MESSAG (n₇+3) = the ID set by the programmer

MESSAG (n_1+4) MESSAG (n_1+n_2-1) contain the relevant block numbers of the entity picked.

MESSAG (n_1+4) holds the block number of the outermost block; MESSAG (n_1+n_2-1) holds the block number of the innermost block (n.b.: $n_2-4 \leq 8$).

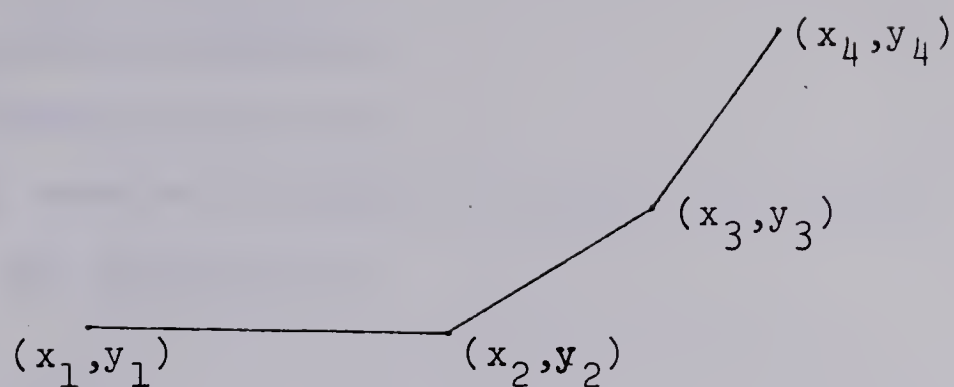
2. Status/function interrupt

MSGTBL (1,I) = 2

$$\text{MSGTBL } (2, I) = n_7 \text{ (say)}$$
$$\text{MSGTBL}(3, I) = 2$$

Then,

MESSAG (n₇) = integer value/or status (0-15)



is drawn, then $n_2 = 8$, and

MESSAG (n_1) = x_1

MESSAG (n_1+1) = y_1

" (n_1+2) = x_2

" (n_1+3) = y_2

" (n_1+4) = x_3

" (n_1+5) = y_3

" (n_1+6) = x_4

" (n_1+7) = y_4

5. Point String

MSGTBL (1,I) = 5

MSGTBL (2,I) = n_1 (say)

MSGTBL (3,I) = n_2 (say)

MESSAG (n_1) to MESSAG (n_1+n_2-1) - an even number of words - gives, in pairs, the coordinates of the series of points.

6. End of Message

MSGTBL (1,I) = 0

Any entries in MSGTBL following the end of message are meaningless.

APPENDIX C*

AN APPLICATIONS PROGRAM USING GRIDSUB

```

      SUBROUTINE GRAFIC
C   ALL APPLICATIONS PROGRAMS ARE TO BE WRITTEN AS
C   SUBROUTINE GRAFIC
C   PROGRAM TO DISPLAY THE EQUATION  $Y=A*X*X+B*X+C$ 
C   POSSIBLE OPERATOR ACTIONS:
C       1. POINT AT COMMAND WORD 'START'
C       2. TYPE IN MAXIMUM X VALUE, A, B, C
C   WHERE X, A, B, C CAN BE +VE OR -VE ONE-DIGIT INTEGERS,
C   OR 1. POINT AT COMMAND WORD 'FINISH'
      INTEGER*2 MSGTBL(3,20),MESSAG(50),CHARS(50)
      INTEGER*2 AX(11),AY(11)
      INTEGER*2 ONE/1/,TWO/2/,THREE/3/
      INTEGER*4 FIRST,GRAPH,FUNC,N1,N2,INT(10)
      REAL Y(11),X(11),Y1(11)
      REAL MAX,MIN
      LOGICAL*1 T/.TRUE./,F/.FALSE./
      COMMON/MESSAGE/MSGTBL,MESSAG
C
C   CONSTRUCT THE DISPLAY BOLCKS
C
      CALL BLOCK(1)
      CALL TEXT(F,0,0,5,'START',T,F)
      CALL ENDBLK
      CALL BLOCK(2)
      CALL TEXT(F,0,0,6,'FINISH',T,F)
      CALL ENDBLK
      CALL BLOCK(3)
      CALL TEXT(F,0,0,26,'FUNCTION -  $A*X*X+B*X+C$ ',T,F)
      CALL TEXT(F,0,-50,38,
1'TO DISPLAY THE FUNCTION,POINT TO START',F,F)
      CALL TEXT(F,0,-100,43,
1'THEN TYPE VALUES FOR X MAXIMUM, A, B, AND C',F,F)
      CALL TEXT(F,0,-150,40,
1'(WITH VALUES -10 TO 10 EXCEPT XMAX > 0 )',F,F)
      CALL TEXT(F,0,-200,35,
1'LEAVING A SPACE BETWEEN EACH NUMBER',F,F)
      CALL MOVE(F,-100,-400)
      CALL ENDBLK
      CALL BLOCK(4)
      CALL MOVE(F,0,50)
      CALL VECTOR(T,12,0,F,F)
      CALL MOVE(F,0,100)
      CALL VECTOR(T,12,0,F,F)
      CALL MOVE(F,0,150)

```



```

CALL VECTOR(T,12,0,F,F)
CALL MOVE(F,0,200)
CALL VECTOR(T,12,0,F,F)
CALL MOVE(F,0,250)
CALL VECTOR(T,12,0,F,F)
CALL MOVE(F,0,300)
CALL VECTOR(T,12,0,F,F)
CALL MOVE(F,0,350)
CALL VECTOR(T,12,0,F,F)
CALL MOVE(F,0,400)
CALL VECTOR(T,12,0,F,F)
CALL MOVE(F,0,450)
CALL VECTOR(T,12,0,F,F)
CALL MOVE(F,0,500)
CALL VECTOR(T,12,0,F,F)
CALL MOVE(F,0,500)
CALL VECTOR(F,0,0,F,F)
CALL MOVE (F,50,0)
CALL VECTOR(T,0,12,F,F)
CALL MOVE(F,100,0)
CALL VECTOR(T,0,12,F,F)
CALL MOVE(F,150,0)
CALL VECTOR(T,0,12,F,F)
CALL MOVE(F,200,0)
CALL VECTOR(T,0,12,F,F)
CALL MOVE(F,250,0)
CALL VECTOR(T,0,12,F,F)
CALL MOVE(F,300,0)
CALL VECTOR(T,0,12,F,F)
CALL MOVE(F,350,0)
CALL VECTOR(T,0,12,F,F)
CALL MOVE(F,400,0)
CALL VECTOR(T,0,12,F,F)
CALL MOVE(F,450,0)
CALL VECTOR(T,0,12,F,F)
CALL MOVE(F,500,0)
CALL VECTOR(T,0,12,F,F)
CALL MOVE(F,500,0)
CALL VECTOR(F,0,0,F,F)
CALL TEXT(F,0,550,1,'Y',T,F)
CALL TEXT(F,550,0,1,'X',T,F)
CALL ENDBLK

```

C

```

CALL DISPLY (1,1,850,800)
CALL DISPLY (2,2,850,700)
CALL DISPLY (3,3,200,650)
FIRST=0
GRAPH=0
FUNC=0
NUM=4

```



```

C
10  CALL TRANMT
    IF(MSGTBL(1,1).NE.CNE) GO TO 21
    N=MSGTBL(2,1)
    IF(MESSAG(N+3).EQ.TWO) STOP
    IF(MESSAG(N+3).NE.ONE) GO TO 22
    IF(MSGTBL(1,2).NE.THREE) GO TO 23
    N1=MSGTBL(2,2)
    N2=MSGTBL(3,2)
    K=N2-2
    IF(N2-2.GT.50) GOTO 50

C
    DO 11 I=1,K
11  CHARS(I)=MESSAG(N1+1+I)
    K=K*2
C  SUBROUTINE CHIN CONVERTS CHARACTERS IN CHARS TO INTEGERS
C  AND STORES THEM IN INT
    CALL CHIN(CHARS,K,NUM,INT)
    XMAX=INT(1)
    A=INT(2)
    B=INT(3)
    C=INT(4)
    HX=XMAX/10.0
    X1=0.0
    DO 1 I =1,11
    Y(I)=(A*X1*X1)+(B*X1)+C
    X(I)=X1
    X1=X1+HX
1  CONTINUE
C
    MAX = Y(1)
    MIN = Y(1)
    DO 4 I=2,11
    IF (Y(I).LT.MIN) MIN=Y(I)
    IF (Y(I).GT.MAX) MAX=Y(I)
4  CONTINUE
    DO 2 I=1,11
    AX(I) = X(I)*100*(5.0/XMAX)
    IF (Y(I).EQ.MIN) GO TO 3
    AY(I) = (Y(I)-MIN)*100*(5.0/(MAX-MIN))
    GO TO 2
3  AY(I) = 0
2  CONTINUE
    IF(FIRST.EQ.0) CALL ERSBK(3)
    FIRST=1
    IF(GRAPH.EQ.0) CALL DISPLY (4,4,170,170)
    GRAPH=1
    IF(FUNC.EQ.1) CALL DELBLK(5)
    CALL BLOCK(5)
    CALL DLINE(F,F,AX,AY,11,F,F)

```



```

      CALL ENDBLK
C
      CALL DISPLY (5,5,170,170)
      FUNC=1
      GOTO 10
21     WRITE(6,201) MSGTBL(1,1)
201    FORMAT(/' FIRST ATTENTION NE LP.  TYPE=',I4)
      STOP
22     WRITE(6,202) MESSAG(N+3)
202    FORMAT(/' LP NOT AT ''START''.  MESSAG=',I4)
      STOP
23     WRITE(6,203) MSGTBL(1,2)
203    FORMAT(/' 2ND ATTENTION NOT A/N.  MSGTBL=',I4)
      STOP
50     WRITE (6,204) K
204    FORMAT(/' NO. OF DIGITS TO BE CONVERTED MORE THAN 50'
1/' NO.='',I4)
      RETURN
      END

```

Alternatively Block 4 representing the axes can be defined as follows:

```

      CALL BLOCK (4)
      DO 10 I=1,10
      CALL MOVE (F,0,I*50)
10     CALL VECTOR (T,12,0,F,F)
      CALL MOVE (F,0,500)
      CALL VECTOR (F,0,0,F,F)
      DO 11 I=1,10
      CALL MOVE (F,I*50,0)
11     CALL VECTOR (T,0,12,F,F)
      CALL MOVE (F,500,0)
      CALL VECTOR (F,0,0,F,F)
      CALL TEXT (F,0,550,1,'Y',T,F)
      CALL TEXT (F,550,0,1,'X',T,F)
      CALL ENDBLK

```

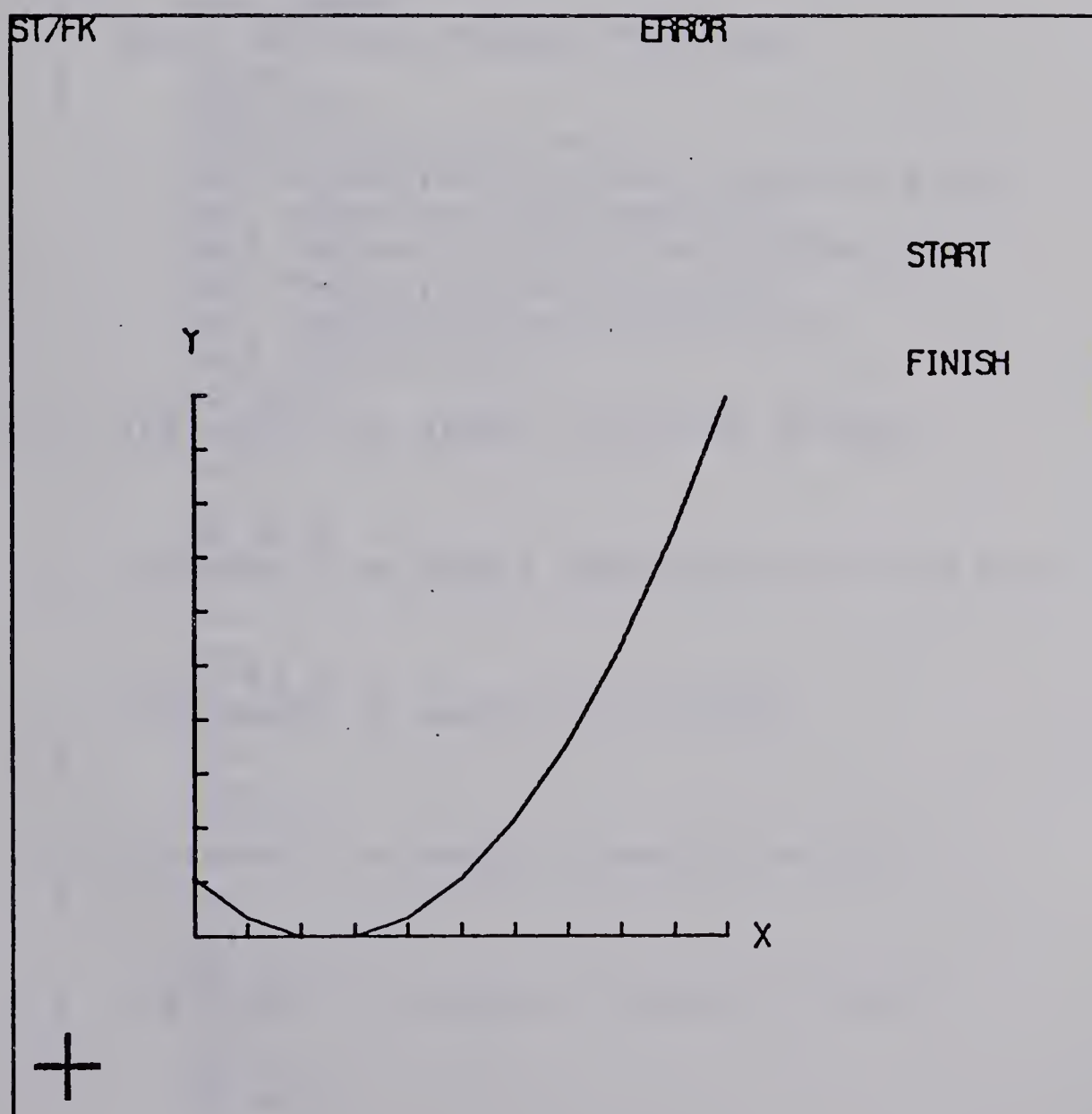

Fig C-1 Display for the applications program
in Appendix C with

$$X \text{ MAX} = 4$$

$$A = 1$$

$$B = -2$$

$$C = 3$$



APPENDIX D*

AN OUTLINE OF AN APPLICATIONS
PROGRAM USING MESSAGE DECODING ROUTINES

```

      SUBROUTINE GRAFIC
C   DEFINE THE VARIABLES TO SERVICE OPERATOR ACTIONS
      IMPLICIT INTEGER (A-Z)
      INTEGER*2 CHAR(43),VX(7),VY(7),PX(20),PY(20)
      INTEGER BLK(8)
C   INITIALIZE AND PREPARE FIRST PICTURE
      .....
C   SEND PICTURE TO GRID
10    CALL TRANMT
C   START DECODING MESSAGE FROM GRID
1     INT=0
2     INT=INT+1
      IF(INT.GT.20) GO TO 9
      CALL DLPEN(INT,LX,LY,MODE,IDENT,BLK,&3)
      CALL DFKEY(INT,STAT,FKEY,&4)
      CALL DALPHA(INT,CX,CY,CN,43,CHAR,&5)
      CALL DVECT(INT,VN,7,VX,VY,&6)
      CALL DPOINT(INT,PN,20,PX,PY,&7)
      CALL DEND(INT,&8)
      GO TO 9
C   STATEMENTS TO HANDLE LIGHT PEN ACTIONS
3     .....
      GO TO 2
C   STATEMENTS TO HANDLE FUNCTION/STATUS ACTIONS
4     .....
      GO TO 2
C   STATEMENTS TO HANDLE A/N STRINGS
5     .....
      GO TO 2
C   STATEMENTS TO HANDLE CONNECTED VECTORS
6     .....
      GO TO 2
C   STATEMENTS TO HANDLE A STRING OF POINTS
7     .....
      GO TO 2

```



```
C  STATEMENTS TO PROCESS END OF MESSAGE
8      ....
      ....
      GO TO 10
C  IN CASE OF BAD SOFTWARE
9      ....
      ....
      RETURN
      END
```

The above is an over-simplified example. In most cases, two or more operator actions are related and must appear in a particular order. The applications programmer is responsible for keeping track of the necessary relationships which are relevant to his application.

B29922